

DMDD ASSIGNMENT 4

TEAM MEMBERS:

VATSAL MEHTA (NUID- 002912412)

VIDIP KAMDAR (NUID- 002701593)

HARSH JAIN (NUID- 002747565)

HRUSHITHA PUTTALA (NUID- 002795117)

GITHUB REPOSITORY LINK:

<https://github.com/Vatsal-77/Smartphone-specification-and-price-classification-system>

Q1. Which brand have what quantity of phones?

use dmdd;

CREATE VIEW group_by AS

SELECT Brand,COUNT(Name) AS No_of_Phones

FROM ndtv_data_final

GROUP BY Brand;

File 19* SQL File 20* SQL File 21* SQL File 22* SQL File 23* SQL File 24* SQL File 25* SQL File 26* self_join group_by x

Limit to 2000 rows

1 • SELECT * FROM dmdd.group_by;

Result Grid Filter Rows: Export: Wrap Cell Content: [fA](#)

	Brand	No_of_Phones
▶	Realme	18
	Apple	17
	LG	42
	OnePlus	11
	Samsung	101
	Asus	37
	Xiaomi	47
	Oppo	35
	Huawei	12
	Google	10
	Nokia	35
	HTC	35
	Motorola	42
	Honor	33
	Yu	13
	Poco	2
	Vivo	52
	Nubia	12
	Black Sh...	1
	Infinix	17
	Lenovo	42
	Sony	28
	Jio	1
	Coolpad	19

group_by 1 x Read Only

Result Grid
Form Editor
Field Types
Query Stats
Execution Plan

Q2. Which model of the smartphone has what clockspeed?

use dmdd;

CREATE VIEW right_join AS

SELECT Model,clock_speed

FROM ndtv_data_final

RIGHT JOIN test

ON ndtv_data_final.Id = test.Id;

The screenshot shows a database query editor with a toolbar at the top. The SQL query entered is: `SELECT * FROM dmdd.right_join;`. Below the query editor, the 'Result Grid' is displayed, showing a table with two columns: 'Model' and 'dock_speed'. The table contains 25 rows of data, listing various smartphone models and their corresponding dock speeds. On the right side of the interface, there is a vertical sidebar with icons for 'Result Grid', 'Form Editor', 'Field Types', 'Query Stats', and 'Execution Plan'. The bottom status bar indicates 'right_join 1' and 'Read Only'.

Model	dock_speed
X2 Pro	1.8
iPhone 11 Pro Max	0.5
iPhone 11	2.8
G8X ThinQ	0.5
7T	1.4
7T Pro	2.9
Galaxy Note 10+	2.4
ROG Phone 2	2.4
Redmi K20 Pro	2.9
K3	0.5
X	2.2
Redmi K20	2.9
7 Pro	1.4
Reno 10x Zoom	2.2
3 Pro	1.8
P30 Pro	1
Redmi Note 7 Pro	0.5
Mate 20 Pro	2.9
V40 ThinQ	1.7
6T	2.6
iPhone XR	0.6
iPhone XS Max	2.6
iPhone XS	1.7
Pixel 3 XL	0.7

Q3. Which brand has what release dates of its phones? (cross join)

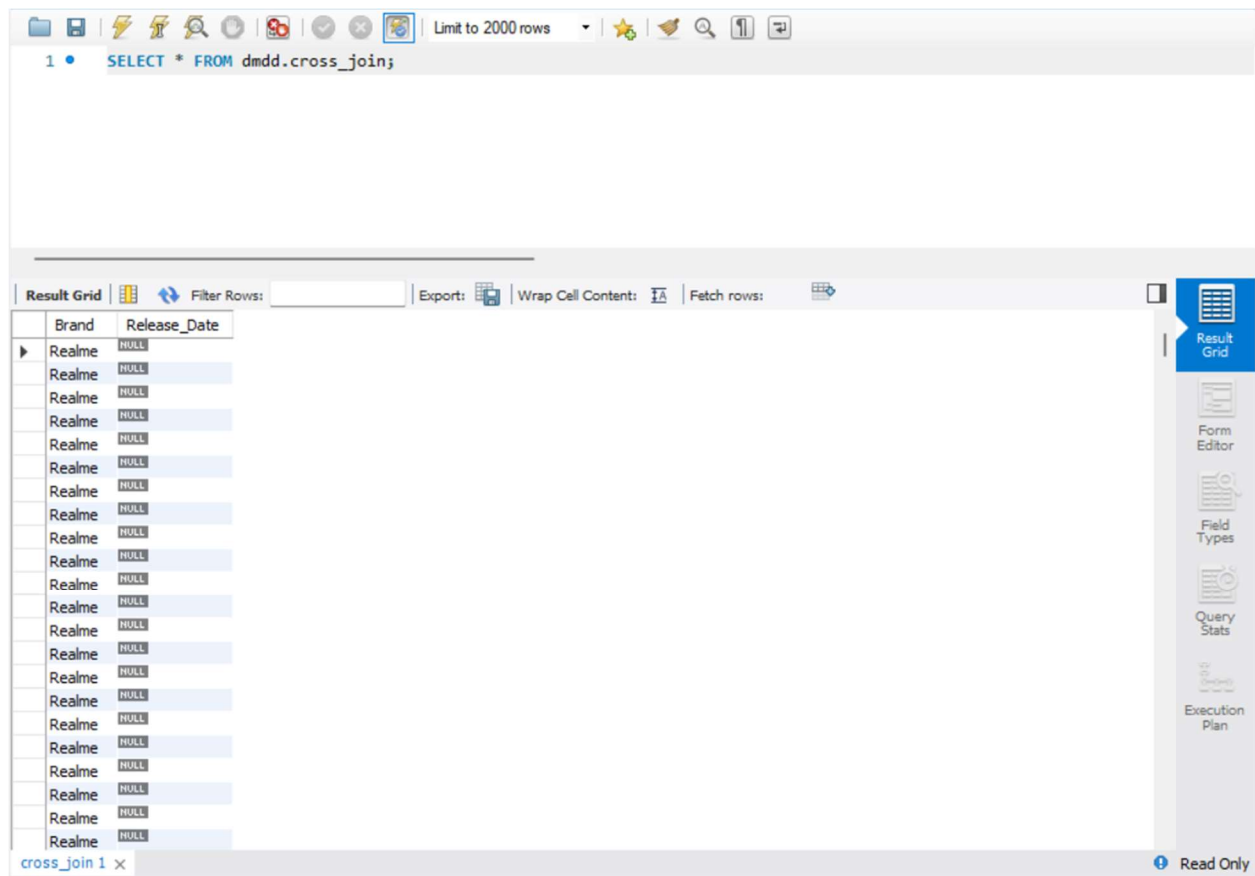
use dmdd;

CREATE VIEW cross_join AS

SELECT ndtv_data_final.Brand, output.release_date AS Release_Date

FROM output

CROSS JOIN dmdd.ndtv_data_final;



Q4. Which smartphone has what weight, pixel height and pixel width?

```
use dmdd;
```

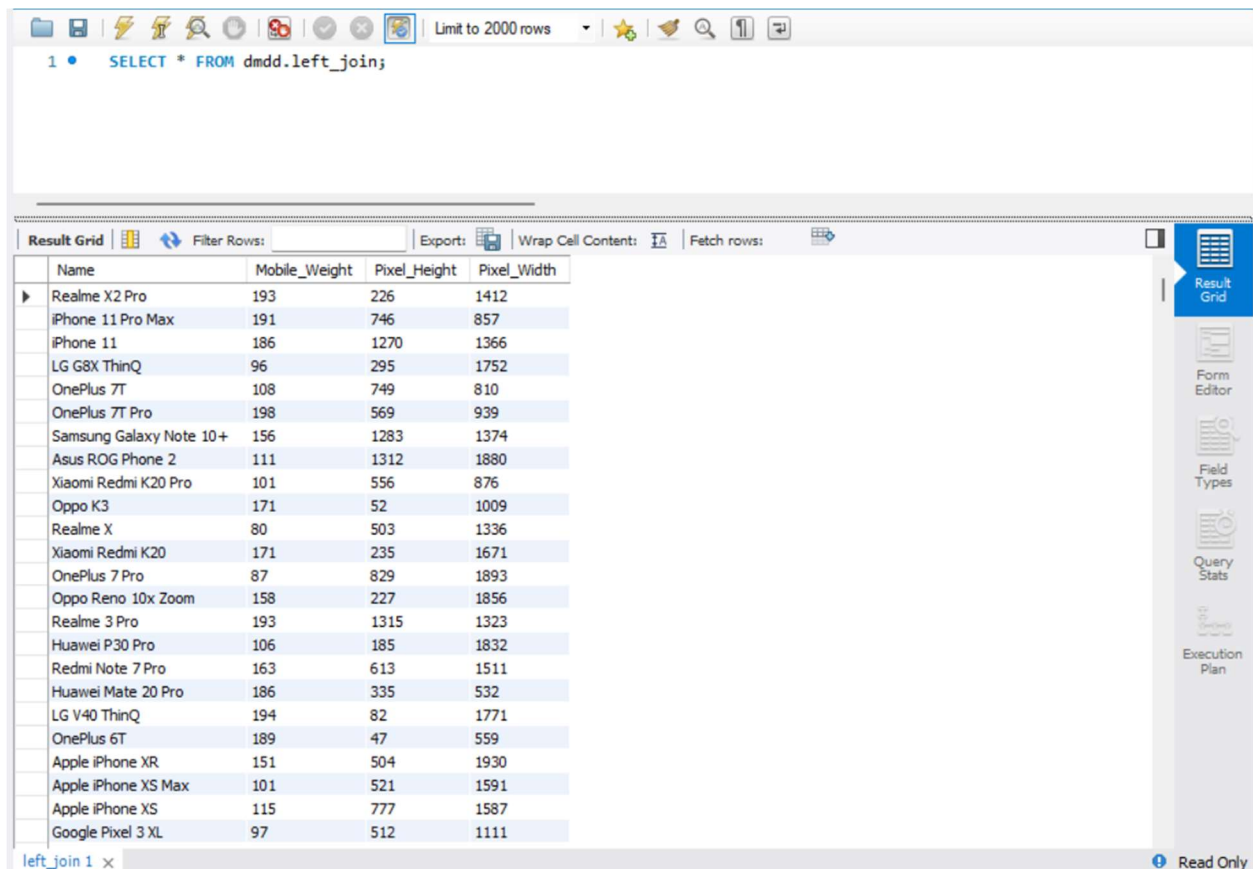
```
CREATE VIEW left_join AS
```

```
SELECT Name,mobile_wt AS Mobile_Weight,px_height As Pixel_Height,px_width
AS Pixel_Width
```

FROM ndtv_data_final

LEFT JOIN test

ON ndtv_data_final.Id = test.Id;



The screenshot shows a database query editor interface. At the top, there's a toolbar with various icons and a text input field containing the SQL query: `1 • SELECT * FROM dmdd.left_join;`. Below the query, there's a section labeled "Result Grid" which displays a table of data. The table has five columns: "Name", "Mobile_Weight", "Pixel_Height", and "Pixel_Width". The data rows list various smartphone models and their corresponding specifications. On the right side of the interface, there's a vertical sidebar with icons for "Result Grid", "Form Editor", "Field Types", "Query Stats", and "Execution Plan". At the bottom right, there's a "Read Only" status indicator.

Name	Mobile_Weight	Pixel_Height	Pixel_Width
Realme X2 Pro	193	226	1412
iPhone 11 Pro Max	191	746	857
iPhone 11	186	1270	1366
LG G8X ThinQ	96	295	1752
OnePlus 7T	108	749	810
OnePlus 7T Pro	198	569	939
Samsung Galaxy Note 10+	156	1283	1374
Asus ROG Phone 2	111	1312	1880
Xiaomi Redmi K20 Pro	101	556	876
Oppo K3	171	52	1009
Realme X	80	503	1336
Xiaomi Redmi K20	171	235	1671
OnePlus 7 Pro	87	829	1893
Oppo Reno 10x Zoom	158	227	1856
Realme 3 Pro	193	1315	1323
Huawei P30 Pro	106	185	1832
Redmi Note 7 Pro	163	613	1511
Huawei Mate 20 Pro	186	335	532
LG V40 ThinQ	194	82	1771
OnePlus 6T	189	47	559
Apple iPhone XR	151	504	1930
Apple iPhone XS Max	101	521	1591
Apple iPhone XS	115	777	1587
Google Pixel 3 XL	97	512	1111

Q5. Which phone model has what release date? (inner join)

use dmdd;

CREATE VIEW inner_join AS

SELECT Model,release_date

FROM ndtv_data_final

INNER JOIN output

ON ndtv_data_final.Id= output.Id;

SQL File 20* SQL File 21* SQL File 22* SQL File 23* SQL File 24* SQL File 25* SQL File 26* having_count inner_join x

Limit to 2000 rows

1 • `SELECT * FROM dmdd.inner_join;`

Result Grid Filter Rows: Export: Wrap Cell Content: `Ctrl+Alt+Z`

Model	release_date
X2 Pro	20-Sep
iPhone 11 Pro Max	20-Sep
iPhone 11	20-Sep
7T	18-Jun
Galaxy Note 10+	19-Jan
ROG Phone 2	21-Jan
Redmi K20 Pro	18-Dec
K3	18-Dec
X	18-Dec
P20 Pro	18-May
6	18-May
ZenFone 5Z (ZS6...	18-May
iPhone 8 Plus	18-Nov
iPhone X	18-Nov
Moto Z2 Force	20-Mar
U11	20-Nov
8	18-Sep
Galaxy S8	20-Sep
G6	20-Oct
3T	20-Oct
Galaxy C9 Pro	20-Oct
iPhone 7	20-Sep
Galaxy A8	19-Sep
X2	20-Sep

inner_join 1 x Read Only

Result Grid
Form Editor
Field Types
Query Stats
Execution Plan

Q6. What is the total number of brands?

use dmdd;

CREATE VIEW union_tables AS

SELECT Brand FROM ndtv_data_final

UNION

SELECT brand_name FROM output;

The screenshot shows a SQL query editor interface. At the top, there is a toolbar with various icons for file operations, execution, and settings. Below the toolbar, the query editor contains the following SQL query:

```
1 • SELECT * FROM dmdd.union_tables;
```

Below the query editor, there is a "Result Grid" section. The grid is currently empty, but it has a header row with the column "Brand". To the right of the grid, there is a sidebar with several icons and labels: "Result Grid", "Form Editor", "Field Types", "Query Stats", and "Execution Plan". At the bottom of the sidebar, there is a "Read Only" status indicator.

Q7. What is the operating system for what number of phones?

use dmdd;

CREATE VIEW having_count AS

SELECT Operating_system, count(Id) AS No_of_Phones

FROM ndtv_data_final

GROUP BY Operating_system

HAVING count(Id) > 5;

The screenshot shows a SQL IDE with multiple tabs. The active tab is 'having_count'. The query editor contains the SQL statement: `SELECT * FROM dmdd.having_count;`. Below the editor, the 'Result Grid' is displayed, showing the following data:

Operating_system	No_of_Phones
Android	1298
iOS	17
Cyanogen	10
BlackBerry	10
Windows	19

The right sidebar contains icons for 'Result Grid', 'Form Editor', 'Field Types', 'Query Stats', and 'Execution Plan'. The status bar at the bottom indicates 'having_count 1 x' and 'Read Only'.

Q8. What is the total number of phones from both the dataset?

use dmdd;

CREATE VIEW union_all AS

SELECT Model From ndtv_data_final

UNION ALL

SELECT model_name FROM output;

The screenshot shows a database query tool interface. At the top, there is a toolbar with various icons and a dropdown menu set to "Limit to 2000 rows". Below the toolbar, the SQL query is displayed: `1 • SELECT * FROM dmdd.union_all;`. The main area shows the "Result Grid" with a list of smartphone models. The models are: X2 Pro, iPhone 11 Pro Max, iPhone 11, G8X ThinQ, 7T, 7T Pro, Galaxy Note 10+, ROG Phone 2, Redmi K20 Pro, K3, X, Redmi K20, 7 Pro, Reno 10x Zoom, 3 Pro, P30 Pro, Redmi Note 7 Pro, Mate 20 Pro, V40 ThinQ, 6T, iPhone XR, iPhone XS Max, iPhone XS, and Pixel 3 XL. The bottom status bar indicates "union_all 1" and "Read Only". On the right side, there is a vertical toolbar with icons for "Result Grid", "Form Editor", "Field Types", "Query Stats", and "Execution Plan".

Model
X2 Pro
iPhone 11 Pro Max
iPhone 11
G8X ThinQ
7T
7T Pro
Galaxy Note 10+
ROG Phone 2
Redmi K20 Pro
K3
X
Redmi K20
7 Pro
Reno 10x Zoom
3 Pro
P30 Pro
Redmi Note 7 Pro
Mate 20 Pro
V40 ThinQ
6T
iPhone XR
iPhone XS Max
iPhone XS
Pixel 3 XL

Q9. What is the average price of all the smartphones?

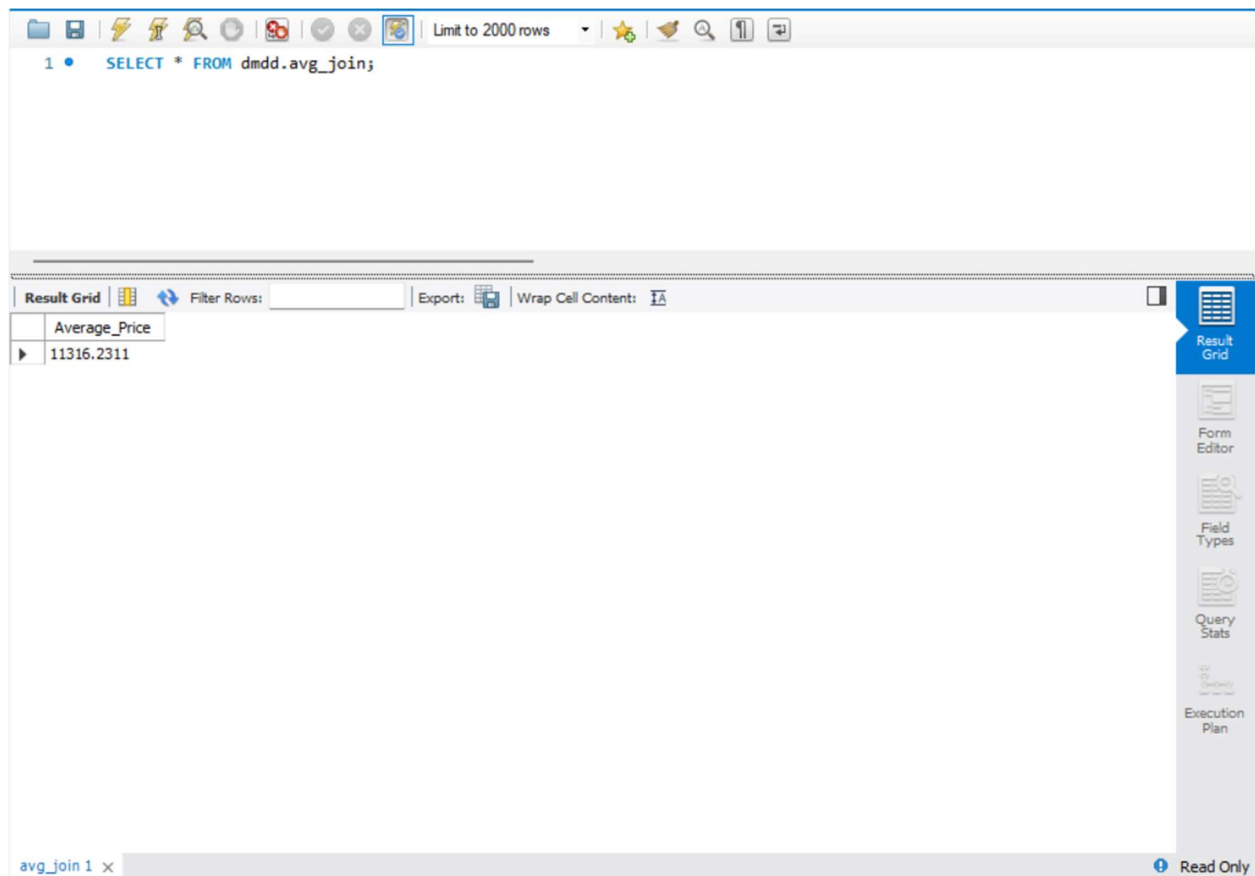
use dmdd;

CREATE VIEW avg_join AS

SELECT AVG(ndtv_data_final.Price) AS Average_Price

FROM ndtv_data_final

INNER JOIN output ON ndtv_data_final.Id = output.Id;



The screenshot shows a SQL query editor interface. At the top, a toolbar contains various icons for file operations, execution, and settings. Below the toolbar, the query text is displayed: `1 • SELECT * FROM dmdd.avg_join;`. A status bar indicates "Limit to 2000 rows". Below the query editor, a "Result Grid" tab is active, showing a single row of results. The column header is "Average_Price" and the value is "11316.2311". To the right of the result grid is a vertical toolbar with icons for "Result Grid", "Form Editor", "Field Types", "Query Stats", and "Execution Plan". At the bottom left, a tab labeled "avg_join 1" is visible. At the bottom right, a "Read Only" status is shown.

Average_Price
11316.2311

Q10. Which brands have what number of phones with internal storage more than 128GB?

use dmdd;

CREATE VIEW where_group_by AS

SELECT Brand,count(Internal_storage) AS Phones_with_higher_storage

FROM ndtv_data_final

WHERE Internal_storage>=128

GROUP BY Brand;

1 • `SELECT * FROM dmdd.where_group_by;`

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

	Brand	Phones_with_higher_storage
▶	LG	4
	OnePlus	5
	Samsung	16
	Asus	4
	Xiaomi	2
	Realme	1
	Oppo	11
	Huawei	6
	HTC	2
	Vivo	9
	Nubia	3
	Motorola	2
	Black Sh...	1
	Honor	6
	Nokia	1
	Infinix	1
	Micromax	1
	Meizu	1
	Panasonic	1
	Tecno	1

Result Grid
Form Editor
Field Types
Query Stats
Execution Plan

dmdd.where_group_by 1 x Read Only

Q11. Which phones have what processor count and core capacity?

use dmdd;

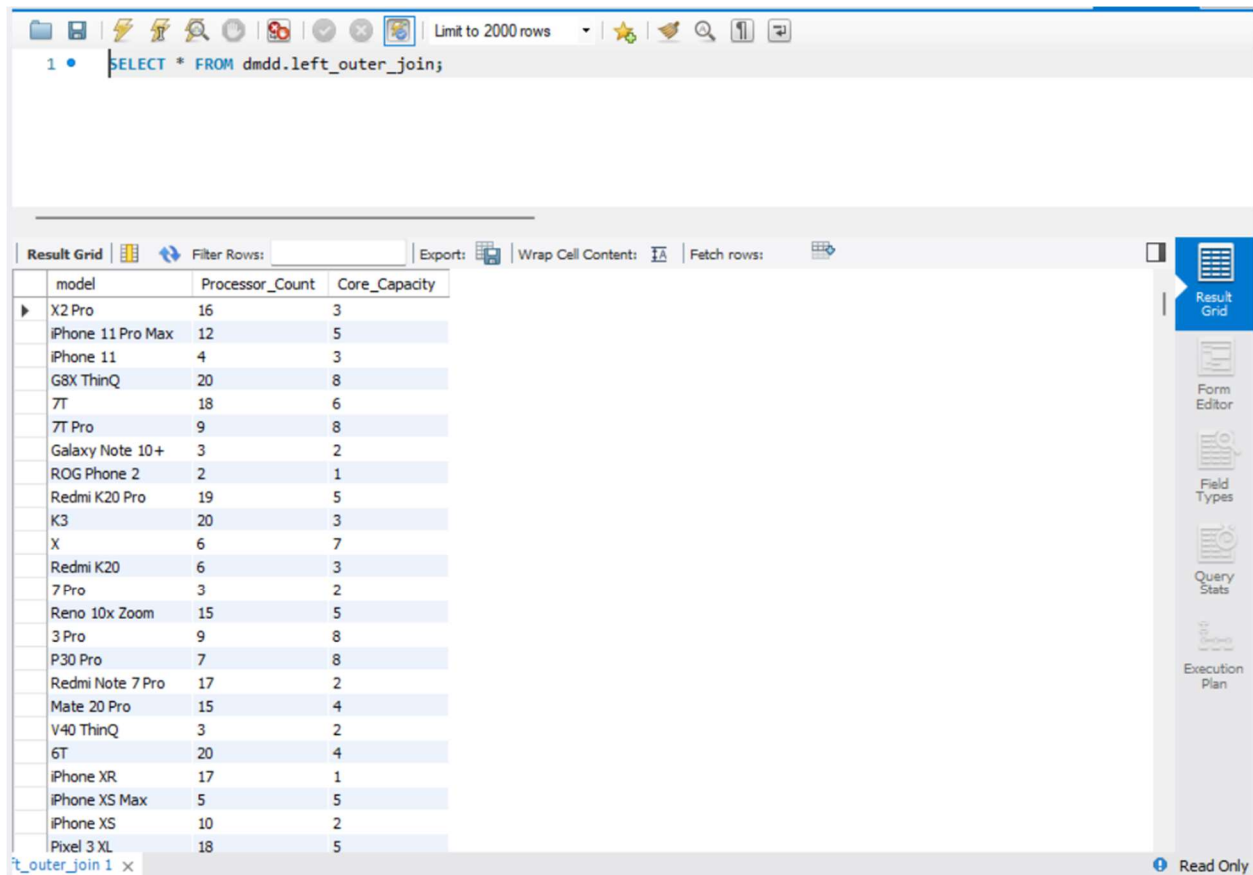
CREATE VIEW left_outer_join AS

SELECT model,pc AS Processor_Count, n_cores AS Core_Capacity

FROM ndtv_data_final

LEFT OUTER JOIN test

ON ndtv_data_final.Id = test.Id



The screenshot shows a database query tool interface. At the top, a SQL query is entered: `SELECT * FROM dmdd.left_outer_join;`. Below the query editor, a toolbar includes options like 'Limit to 2000 rows', 'Export', 'Wrap Cell Content', and 'Fetch rows'. The main area displays a 'Result Grid' with a table of data. The table has three columns: 'model', 'Processor_Count', and 'Core_Capacity'. It lists various smartphone models and their specifications. On the right side, there is a vertical toolbar with icons for 'Result Grid', 'Form Editor', 'Field Types', 'Query Stats', and 'Execution Plan'. At the bottom right, a 'Read Only' status is indicated.

model	Processor_Count	Core_Capacity
X2 Pro	16	3
iPhone 11 Pro Max	12	5
iPhone 11	4	3
GSX ThinQ	20	8
7T	18	6
7T Pro	9	8
Galaxy Note 10+	3	2
ROG Phone 2	2	1
Redmi K20 Pro	19	5
K3	20	3
X	6	7
Redmi K20	6	3
7 Pro	3	2
Reno 10x Zoom	15	5
3 Pro	9	8
P30 Pro	7	8
Redmi Note 7 Pro	17	2
Mate 20 Pro	15	4
V40 ThinQ	3	2
6T	20	4
iPhone XR	17	1
iPhone XS Max	5	5
iPhone XS	10	2
Pixel 3 XL	18	5

Q12. What is the highest and lowest price of the phones of each brand?

use dmdd;

CREATE VIEW right_outer_join AS

SELECT output.Id, Name, Brand, highest_price, lowest_price

FROM ndtv_data_final

RIGHT OUTER JOIN output

ON ndtv_data_final.Id = output.Id

Limit to 2000 rows

1 • `SELECT * FROM dmdd.right_outer_join;`

Result Grid Filter Rows: Export: Wrap Cell Content: [f](#)

Id	Name	Brand	highest_price	lowest_price
1	Realme X2 Pro	Realme	2489	1659
2	iPhone 11 Pro Max	Apple	2489	1659
3	iPhone 11	Apple	2489	1659
5	OnePlus 7T	OnePlus	11099	10631
7	Samsung Galaxy Note 10+	Samsung	5295	4733
8	Asus ROG Phone 2	Asus	5222	4990
9	Xiaomi Redmi K20 Pro	Xiaomi	5372	4646
10	Oppo K3	Oppo	5372	4646
11	Realme X	Realme	5559	4897
30	Huawei P20 Pro	Huawei	5448	4970
31	OnePlus 6	OnePlus	5448	4970
32	Asus ZenFone 5Z (ZS620KL)	Asus	5448	4970
40	Apple iPhone 8 Plus	Apple	6526	5290
41	Apple iPhone X	Apple	5759	4599
42	Motorola Moto Z2 Force	Motorola	5699	5599
43	HTC U11	HTC	7786	6840
44	Nokia 8	Nokia	8196	7049
46	Samsung Galaxy S8	Samsung	3185	2168
47	LG G6	LG	3299	2889
48	OnePlus 3T	OnePlus	3899	3581
49	Samsung Galaxy C9 Pro	Samsung	3899	3581
52	Apple iPhone 7	Apple	3299	2999
63	Samsung Galaxy A8	Samsung	2689	2395
68	Poco X2	Poco	4049	3550

Result Grid Form Editor Field Types Query Stats Execution Plan

Read Only

Q13. What is the name, popularity and battery power for each mobile phone where popularity is greater than 500?

use dmdd;

CREATE VIEW triple_join AS

SELECT Name, popularity, battery_power

FROM ndtv_data_final

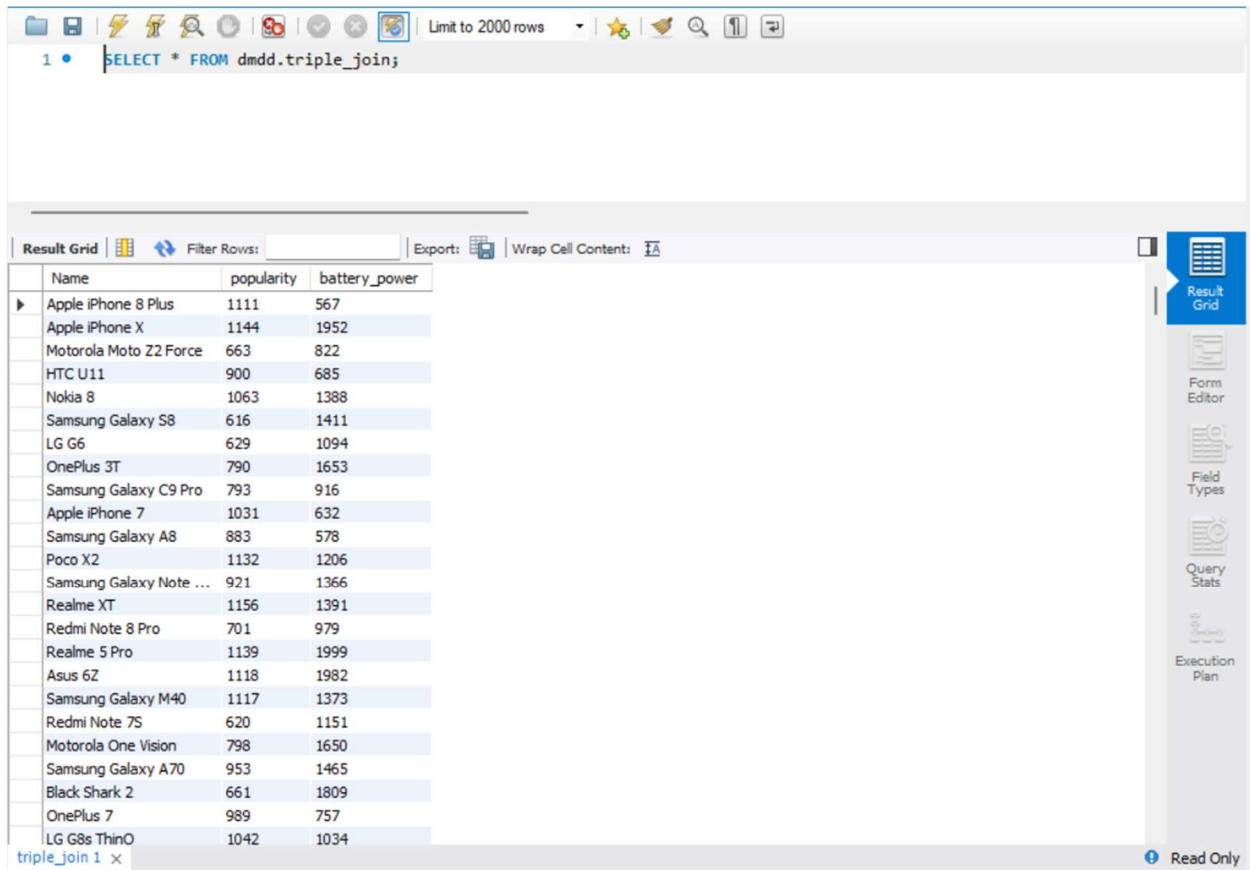
JOIN output

ON ndtv_data_final.Id = output.Id

JOIN test

ON output.Id = test.Id

WHERE popularity>500;



The screenshot shows a database query tool interface. At the top, a toolbar contains icons for file operations, search, and execution. Below the toolbar, a SQL query is entered in a text area: `1 • SELECT * FROM dmdd.triple_join;`. The query is executed, and the results are displayed in a table. The table has three columns: 'Name', 'popularity', and 'battery_power'. The results list various smartphone models with their respective popularity scores and battery power values. The table is titled 'triple_join 1' and is in 'Read Only' mode.

Name	popularity	battery_power
Apple iPhone 8 Plus	1111	567
Apple iPhone X	1144	1952
Motorola Moto Z2 Force	663	822
HTC U11	900	685
Nokia 8	1063	1388
Samsung Galaxy S8	616	1411
LG G6	629	1094
OnePlus 3T	790	1653
Samsung Galaxy C9 Pro	793	916
Apple iPhone 7	1031	632
Samsung Galaxy A8	883	578
Poco X2	1132	1206
Samsung Galaxy Note ...	921	1366
Realme XT	1156	1391
Redmi Note 8 Pro	701	979
Realme 5 Pro	1139	1999
Asus 6Z	1118	1982
Samsung Galaxy M40	1117	1373
Redmi Note 7S	620	1151
Motorola One Vision	798	1650
Samsung Galaxy A70	953	1465
Black Shark 2	661	1809
OnePlus 7	989	757
LG G8s ThinO	1042	1034

Q14. Which phones have the mobile weight less than 100?

use dmdd;

CREATE VIEW where_exists AS

SELECT Id,Name

FROM ndtv_data_final

WHERE EXISTS (SELECT mobile_wt FROM test WHERE ndtv_data_final.Id = test.Id
AND mobile_wt < 100);

Limit to 2000 rows

1 • `SELECT * FROM dmdd.where_exists;`

	Id	Name
▶	4	LG G8X ThinQ
	11	Realme X
	13	OnePlus 7 Pro
	24	Google Pixel 3 XL
	28	LG G7+ ThinQ
	30	Huawei P20 Pro
	37	HTC U11+
	40	Apple iPhone 8 Plus
	46	Samsung Galaxy S8
	60	Google Nexus 6P
	61	Apple iPhone 6s Plus
	66	Xiaomi Redmi Note
	89	Asus 6Z
	90	Samsung Galaxy M40
	92	Motorola One Vision
	96	Samsung Galaxy A70
	100	Vivo V15 Pro
	103	Samsung Galaxy M20
	108	Realme U1
	123	Nokia 6.1 Plus
	127	Realme 1
	136	Infinix Zero 5 Pro
	139	Honor 9i
	147	Google Pixel 2 XL

where_exists 1 x

Read Only

Q15. The brand name APPLE has what number of sellers?

use dmdd;

CREATE VIEW where_less_than AS

SELECT ndtv_data_final.Id, Brand, sellers_amount AS No_of_sellers

FROM ndtv_data_final

JOIN output

ON ndtv_data_final.Id = output.Id

where ndtv_data_final.Brand='Apple' and output.sellers_amount<20;

The screenshot shows a database query tool interface. At the top, there is a toolbar with various icons and a text input field containing the SQL query: `1 • SELECT * FROM dmdd.where_less_than;`. Below the query, there is a section for the results. On the left, there is a 'Result Grid' tab. The grid displays the following data:

	Id	Brand	No_of_sellers
▶	40	Apple	11
	41	Apple	19
	52	Apple	17
	192	Apple	10

On the right side of the interface, there is a vertical toolbar with icons for 'Result Grid', 'Form Editor', 'Field Types', 'Query Stats', and 'Execution Plan'. At the bottom right, there is a 'Read Only' status indicator.

Q16. Which of the following mobile phones are black in colour with their prices above 40000?

use dmdd;

CREATE VIEW where_like AS

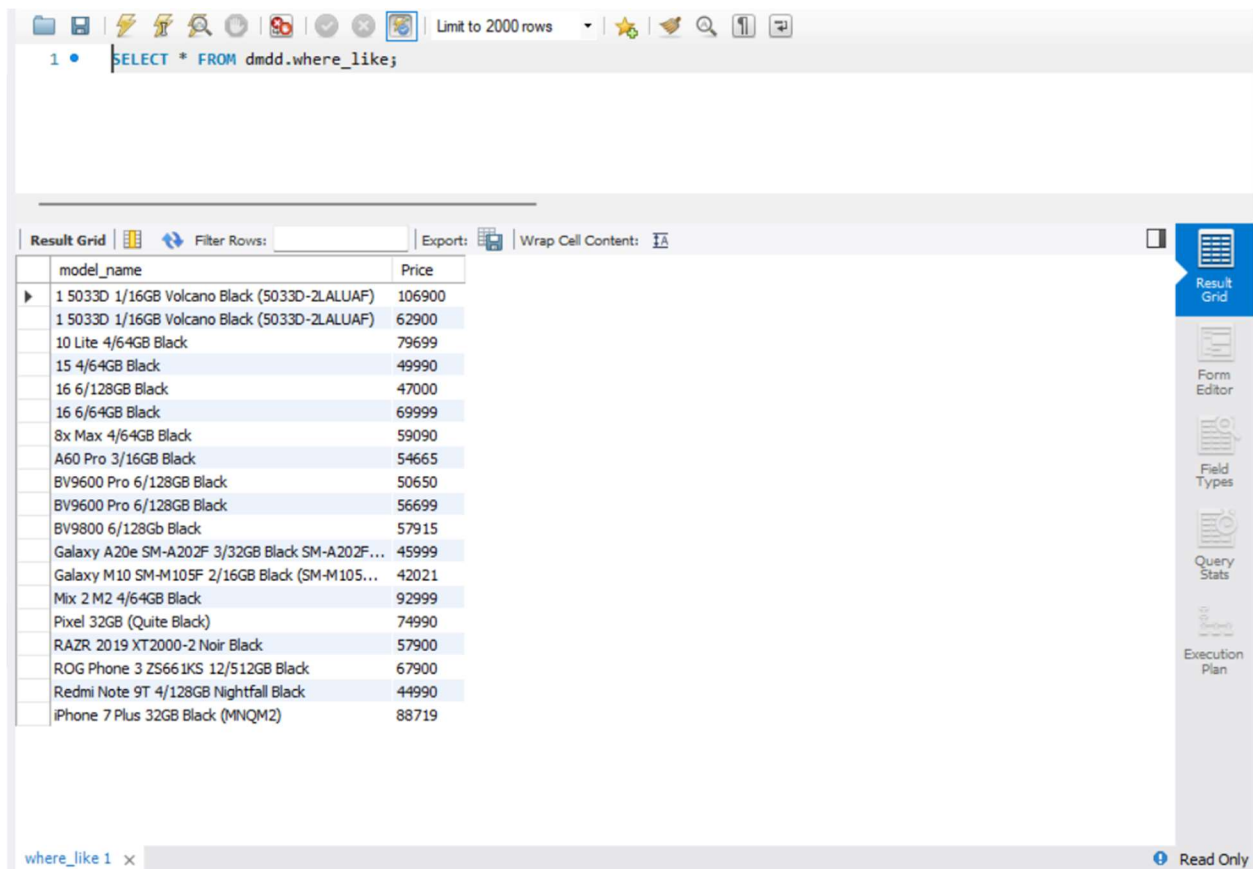
SELECT model_name, Price

FROM ndtv_data_final

JOIN output

ON ndtv_data_final.Id = output.Id

where output.model_name like '%Black%' and ndtv_data_final.Price>40000;



The screenshot shows a database query interface. At the top, a SQL query is entered: `SELECT * FROM dmdd.where_like;`. Below the query, a table of results is displayed. The table has two columns: `model_name` and `Price`. The results list various smartphone models and their prices. The interface includes a toolbar with icons for saving, undo, redo, and other functions. On the right side, there is a sidebar with options like 'Result Grid', 'Form Editor', 'Field Types', 'Query Stats', and 'Execution Plan'. The bottom status bar shows 'where_like 1' and 'Read Only'.

model_name	Price
1 5033D 1/16GB Volcano Black (5033D-2LALUAF)	106900
1 5033D 1/16GB Volcano Black (5033D-2LALUAF)	62900
10 Lite 4/64GB Black	79699
15 4/64GB Black	49990
16 6/128GB Black	47000
16 6/64GB Black	69999
8x Max 4/64GB Black	59090
A60 Pro 3/16GB Black	54665
BV9600 Pro 6/128GB Black	50650
BV9600 Pro 6/128GB Black	56699
BV9800 6/128Gb Black	57915
Galaxy A20e SM-A202F 3/32GB Black SM-A202F...	45999
Galaxy M10 SM-M105F 2/16GB Black (SM-M105...	42021
Mix 2 M2 4/64GB Black	92999
Pixel 32GB (Quite Black)	74990
RAZR 2019 XT2000-2 Noir Black	57900
ROG Phone 3 ZS661KS 12/512GB Black	67900
Redmi Note 9T 4/128GB Nightfall Black	44990
iPhone 7 Plus 32GB Black (MNQM2)	88719

Q17. Brand name, Operating systems and model name of one of the dataset were inserted into another dataset with similar attributes whose price is greater than 10000

use dmdd;

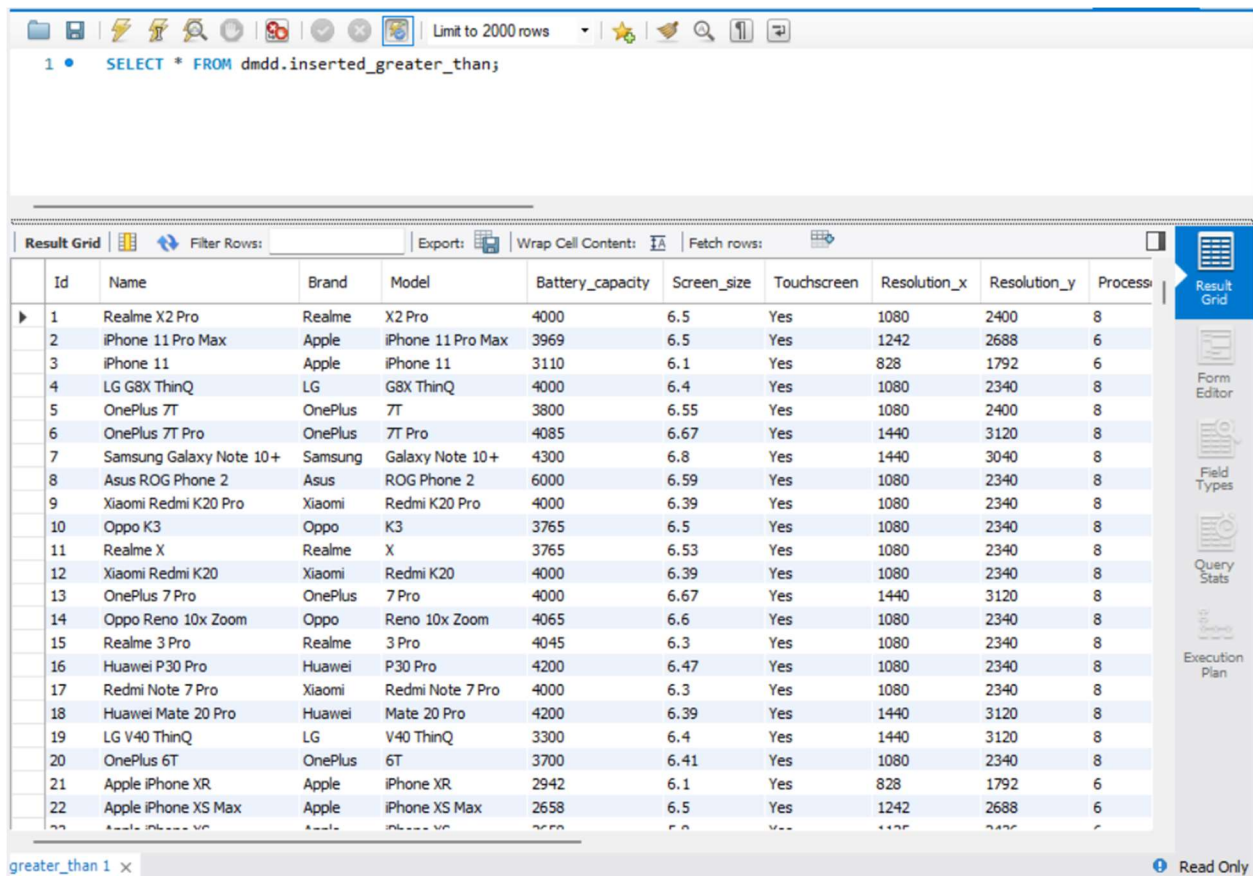
```
INSERT INTO ndtv_data_final(Brand,Model,Operating_system)
```

```
SELECT brand_name, model_name,os FROM output
```

```
WHERE highest_price>10000;
```

```
CREATE VIEW inserted_greater_than AS
```

```
SELECT * FROM dmdd.ndtv_data_final;
```



The screenshot shows a database query tool interface. At the top, there's a toolbar with various icons and a text input field containing the SQL query: `SELECT * FROM dmdd.inserted_greater_than;`. Below the query, there's a "Result Grid" section. The grid has columns: Id, Name, Brand, Model, Battery_capacity, Screen_size, Touchscreen, Resolution_x, Resolution_y, and Process. The data is displayed in a table with 22 rows. The right sidebar contains icons for "Result Grid", "Form Editor", "Field Types", "Query Stats", and "Execution Plan". At the bottom right, there's a "Read Only" indicator.

Id	Name	Brand	Model	Battery_capacity	Screen_size	Touchscreen	Resolution_x	Resolution_y	Process
1	Realme X2 Pro	Realme	X2 Pro	4000	6.5	Yes	1080	2400	8
2	iPhone 11 Pro Max	Apple	iPhone 11 Pro Max	3969	6.5	Yes	1242	2688	6
3	iPhone 11	Apple	iPhone 11	3110	6.1	Yes	828	1792	6
4	LG G8X ThinQ	LG	G8X ThinQ	4000	6.4	Yes	1080	2340	8
5	OnePlus 7T	OnePlus	7T	3800	6.55	Yes	1080	2400	8
6	OnePlus 7T Pro	OnePlus	7T Pro	4085	6.67	Yes	1440	3120	8
7	Samsung Galaxy Note 10+	Samsung	Galaxy Note 10+	4300	6.8	Yes	1440	3040	8
8	Asus ROG Phone 2	Asus	ROG Phone 2	6000	6.59	Yes	1080	2340	8
9	Xiaomi Redmi K20 Pro	Xiaomi	Redmi K20 Pro	4000	6.39	Yes	1080	2340	8
10	Oppo K3	Oppo	K3	3765	6.5	Yes	1080	2340	8
11	Realme X	Realme	X	3765	6.53	Yes	1080	2340	8
12	Xiaomi Redmi K20	Xiaomi	Redmi K20	4000	6.39	Yes	1080	2340	8
13	OnePlus 7 Pro	OnePlus	7 Pro	4000	6.67	Yes	1440	3120	8
14	Oppo Reno 10x Zoom	Oppo	Reno 10x Zoom	4065	6.6	Yes	1080	2340	8
15	Realme 3 Pro	Realme	3 Pro	4045	6.3	Yes	1080	2340	8
16	Huawei P30 Pro	Huawei	P30 Pro	4200	6.47	Yes	1080	2340	8
17	Redmi Note 7 Pro	Xiaomi	Redmi Note 7 Pro	4000	6.3	Yes	1080	2340	8
18	Huawei Mate 20 Pro	Huawei	Mate 20 Pro	4200	6.39	Yes	1440	3120	8
19	LG V40 ThinQ	LG	V40 ThinQ	3300	6.4	Yes	1440	3120	8
20	OnePlus 6T	OnePlus	6T	3700	6.41	Yes	1080	2340	8
21	Apple iPhone XR	Apple	iPhone XR	2942	6.1	Yes	828	1792	6
22	Apple iPhone XS Max	Apple	iPhone XS Max	2658	6.5	Yes	1242	2688	6

Q18. Brand name, Operating systems and model name of one of the dataset were inserted into another dataset with similar attributes whose brand name is OnePlus

use dmdd;

```
INSERT INTO output(brand_name,model_name,screen_size,best_price)
```

```
SELECT Brand, Model, Screen_size, Price FROM ndtv_data_final
```

```
WHERE Brand like 'OnePlus';
```

```
CREATE VIEW inserted_like AS
```

```
SELECT * FROM output;
```

```
1 • SELECT * FROM dmdd.inserted_like;
```

Result Grid									
Filter Rows: Export: Wrap Cell Content:									
	Id	brand_name	model_name	os	popularity	best_price	lowest_price	highest_price	sellers_amount
▶	1	ALCATEL	1 5033D 1/16GB Volcano Black (5033D-2LALUAF)	Android	323	1803	1659	2489	36
	2	ALCATEL	1 5033D 1/16GB Volcano Black (5033D-2LALUAF)	Android	299	1803	1659	2489	36
	3	ALCATEL	1 5033D 1/16GB Volcano Black (5033D-2LALUAF)	Android	287	1803	1659	2489	36
	5	Honor	10 6/64GB Black	Android	71	10865	10631	11099	2
	7	Honor	10 Lite 4/64GB Black	Android	134	4973	4733	5295	6
	8	Honor	10 lite 3/128GB Blue	Android	477	5100	4990	5222	3
	9	Honor	10 lite 3/64GB Black	Android	215	4948	4646	5372	8
	10	Honor	10 lite 3/64GB Black	Android	179	4948	4646	5372	8
	11	Honor	10 lite 3/64GB Blue	Android	437	5165	4897	5559	7
	30	Meizu	15 4/64GB Black	Android	166	5272	4970	5448	3
	31	Meizu	15 4/64GB Black	Android	125	5272	4970	5448	3
	32	Meizu	15 4/64GB Black	Android	116	5272	4970	5448	3
	40	Meizu	16 6/128GB Black	Android	1111	5926	5290	6526	11
	41	Meizu	16 6/64GB Black	Android	1144	5231	4599	5759	19
	42	Meizu	16X 6/128GB Dual Purple	Android	663	5649	5599	5699	2
	43	Meizu	16Xs 6/128GB Pearl White	Android	900	7229	6840	7786	11
	44	Meizu	16th 6/64GB Black	Android	1063	7776	7049	8196	9
	46	ALCATEL	1B 5002H Prime Black (5002H-2AALUA12)	Android	616	2408	2168	3185	31
	47	ALCATEL	1SE 3/32GB Power Gray (5030D-2AALUA2)	Android	629	3249	2889	3299	38
	48	ALCATEL	1SE 4/128GB Agate Green (5030E-2BALUA2)	Android	790	3877	3581	3899	35
	49	ALCATEL	1SE 4/128GB Agate Green (5030E-2BALUA2)	Android	793	3877	3581	3899	35
	52	Nokia	2.4 2/32GB Charcoal	Android	1031	3162	2999	3299	17
	63	Nokia	2720 Flip Black (16BTSB01A10)	iOS	883	2530	2395	2689	26

inserted_like 1 x

Read Only

Q19. Which of the following mobile phones have following screen size, where the quantity is above 10?

use dmdd;

CREATE VIEW group_by_having AS

SELECT Screen_size, count(Id) AS No_of_Phones

FROM ndtv_data_final

GROUP BY Screen_size

HAVING count(Id) > 10;

1 • `SELECT * FROM dmdd.group_by_having;`

Limit to 2000 rows

Result Grid | Filter Rows: | Export: | Wrap Cell Content: `IA`

	Screen_size	No_of_Phones
▶	6.4	21
	6.3	28
	5.5	251
	6	53
	5.99	14
	6.2	23
	4.7	37
	5.7	51
	5.2	82
	5	404
	4	71
	4.5	80
	6.22	15
	5.45	25

p_by_having 1 x Read Only

Q20. Self joined brands from both the tables

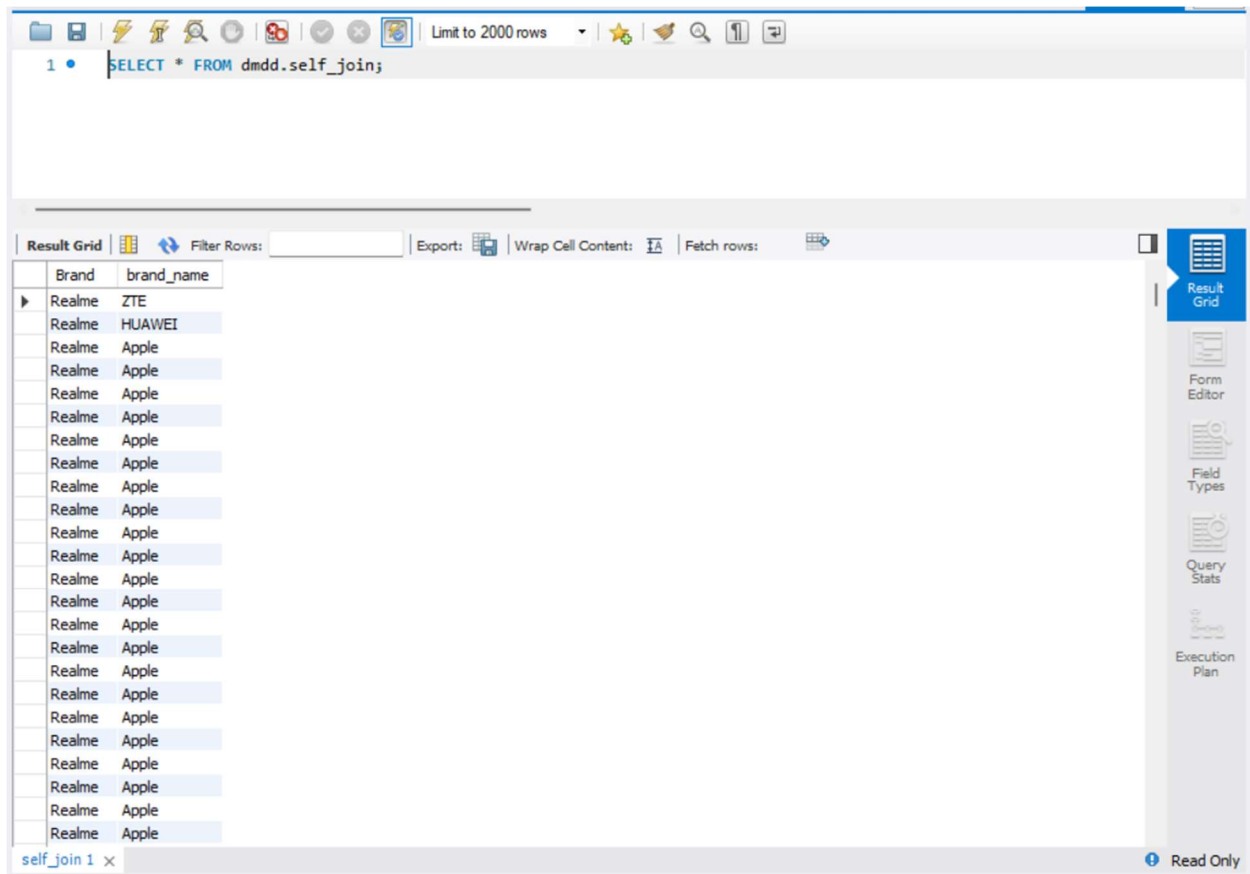
use dmdd;

CREATE VIEW self_join AS

SELECT ndtv_data_final.Brand, output.brand_name

FROM ndtv_data_final, output

WHERE ndtv_data_final.Id <> output.Id



Normalization:

1. 1NF:


First Normal Form: Satisfies 1NF requirements

The table has a primary key attribute identified as Id

No multi-value attributes present

No repeating groups







Function:



Name:

The name of the routine is parsed automatically from the DDL statement. The DDL is parsed automatically while you type.

DDL:




```
1 • CREATE DEFINER=`root`@`localhost` FUNCTION `new_function`(a int) RETURNS varchar(20) CHARSET utf8
2     READS SQL DATA
3     DETERMINISTIC
4     BEGIN
5         declare result varchar(20);
6         SET global log_bin_trust_function_creators=1;
7         IF a>0 THEN
8             SET result = "table is not in 1NF";
9         ELSE
10            SET result = 'table in 1 NF';
11        END IF;
12        RETURN result;
13    END
```

Routine

Apply

Revert







Procedure:



Name:

The name of the routine is parsed automatically from the DDL statement. The DDL is parsed automatically while you type.

DDL:






































```
1 • CREATE DEFINER='root'@'localhost' PROCEDURE `new_procedure` (out var int)
2 • BEGIN
3 •     select count(*) into var from dmdd.ndtv_data_final where
4 •     Model like '%,%';
5 • END
```


















Routine

Apply

Revert



Limit to 2000 rows



```
1 • use dmdd;
2
3 • set @var = 0;
4 • call ndtv_data_final.new_procedure(@var);
5 • select @var;
6
7 • select dmdd.new_function(@var);
```


Outputs:

The screenshot displays a SQL IDE interface. At the top, a toolbar includes icons for 'Result Grid', 'Filter Rows', 'Export', and 'Wrap Cell Content'. Below this, a small table shows the variable '@var' with a value of 1.

The main query editor contains the following SQL script:

```
1 • use dmdd;  
2  
3 • set @var = 0;  
4 • call ndtv_data_final.new_procedure(@var);  
5 • select @var;  
6  
7 • select dmdd.new_function(@var);
```

Below the query editor, another 'Result Grid' toolbar is visible. The result table below it shows the output of the final query:

dmdd.new_function(@var)
table in 1 NF

On the right side of the IDE, a vertical toolbar contains icons for 'Result Grid', 'Form Editor', 'Field Types', 'Query Stats', and 'Execution Plan'. At the bottom left, a tab labeled 'Result 10' is active. At the bottom right, a 'Read Only' status indicator is present.

2. 2NF:

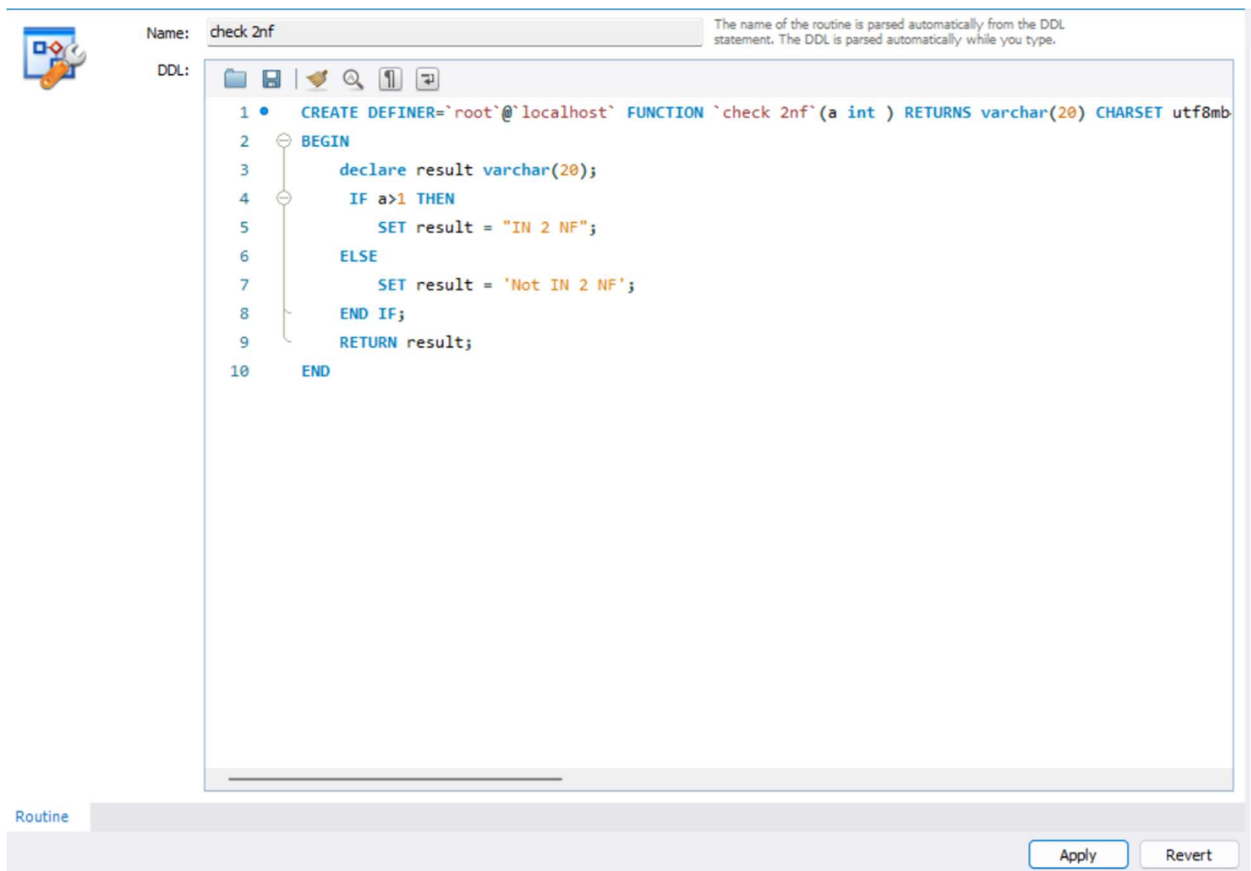
Satisfies 2NF requirements

All requirements for 1NF met

No partial dependency on any column

None of the fields have data calculated from other fields

Function:

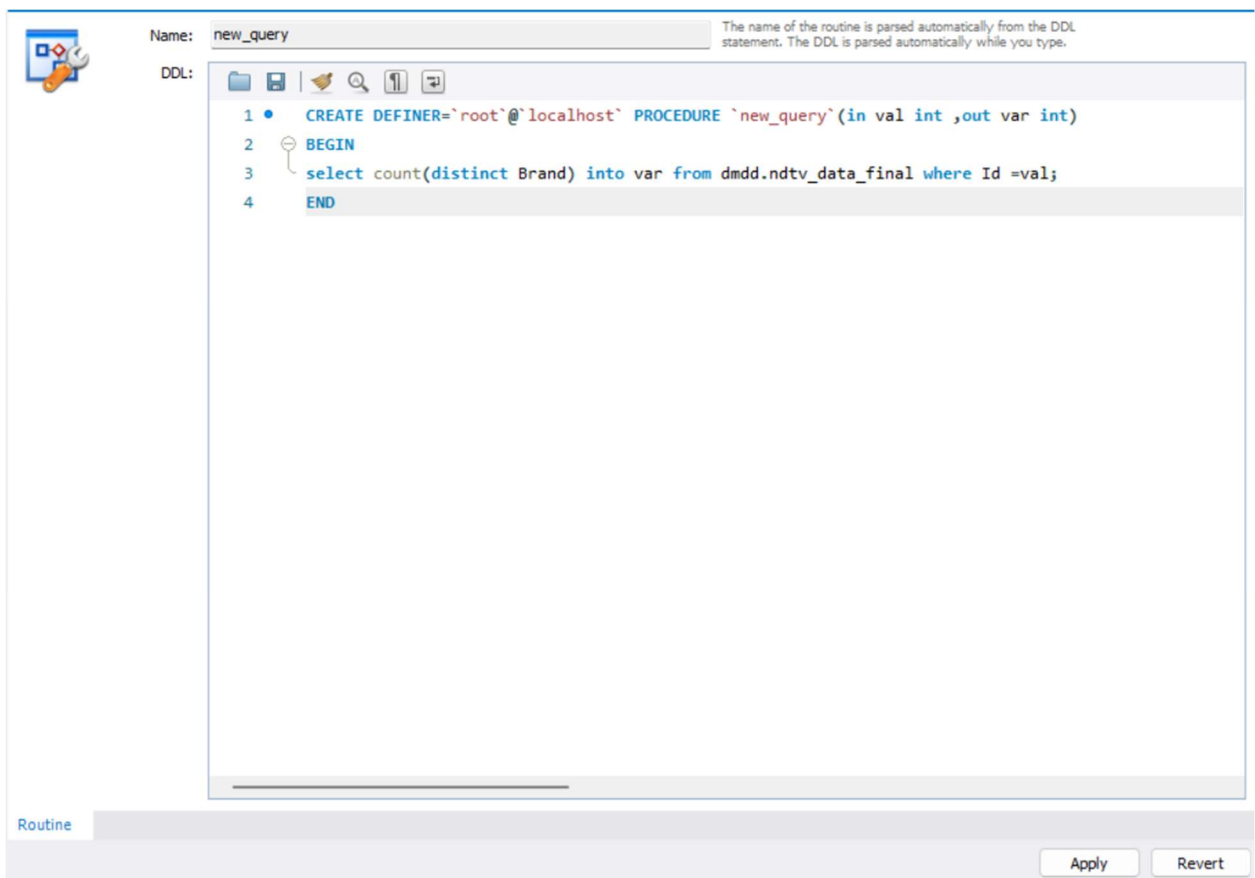


The screenshot shows the MySQL Workbench interface. At the top, the 'Name' field is set to 'check 2nf'. Below it, the 'DDL' tab is active, displaying the following SQL code:

```
1 CREATE DEFINER='root'@'localhost' FUNCTION `check 2nf` (a int ) RETURNS varchar(20) CHARSET utf8mb4  
2 BEGIN  
3     declare result varchar(20);  
4     IF a>1 THEN  
5         SET result = "IN 2 NF";  
6     ELSE  
7         SET result = 'Not IN 2 NF';  
8     END IF;  
9     RETURN result;  
10 END
```

At the bottom of the window, there is a 'Routine' tab and two buttons: 'Apply' and 'Revert'.


Procedure:



The screenshot shows a database management interface. At the top, there is a "Name:" field containing "new_query". To the right of this field, a small text box states: "The name of the routine is parsed automatically from the DDL statement. The DDL is parsed automatically while you type." Below the name field is a "DDL:" label followed by a toolbar with icons for file operations (folder, save, copy, search, undo, redo). The main area is a text editor containing the following SQL code:


```
1 • CREATE DEFINER='root'@'localhost' PROCEDURE `new_query`(in val int ,out var int)
2 BEGIN
3 select count(distinct Brand) into var from dmdd.ndtv_data_final where Id =val;
4 END
```

At the bottom left, there is a tab labeled "Routine". At the bottom right, there are two buttons: "Apply" and "Revert".



Name: The name of the routine is parsed automatically from the DDL statement. The DDL is parsed automatically while you type.


DDL:





```
1 • CREATE DEFINER='root'@'localhost' PROCEDURE `new_query` (in val int ,out var int)
2 BEGIN
3 select count(distinct Brand) into var from dmdd.ndtv_data_final where Id =val;
4 END
```


Routine

Outputs:

Result Grid 

 Filter Rows:

Export: 

Wrap Cell Content: 

	@var
▶	1

The screenshot shows a database query editor interface. The top toolbar includes icons for file operations, execution, and a 'Limit to 2000 rows' dropdown. The SQL script in the editor is as follows:

```
1 • set @var = 0;
2 • call dmdd.new_query(6,@var);
3 • select @var;
4
5 • select dmdd.`check 2nf`(6);
6
7
```

Below the script, the 'Result Grid' tab is active, displaying the results of the query. The results are shown in a table with two columns: the first column contains the query text 'dmdd.`check 2nf`(6)' and the second column contains the result 'IN 2 NF'.

dmdd.`check 2nf`(6)	
	IN 2 NF

The right sidebar contains several utility icons: 'Result Grid' (active), 'Form Editor', 'Field Types', 'Query Stats', and 'Execution Plan'. The bottom status bar indicates 'Result 29' and 'Read Only'.

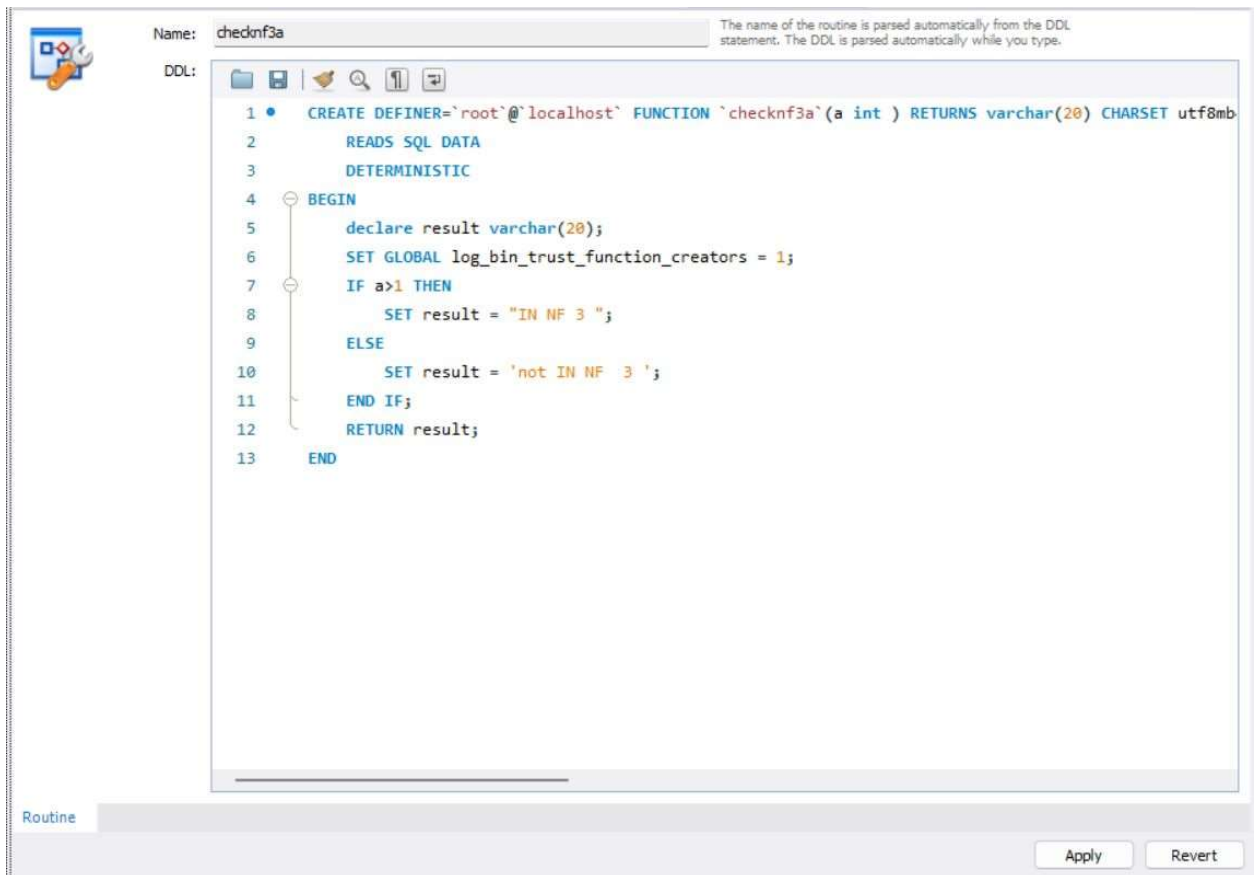
3. 3NF:

Satisfies 3NF requirements

2NF requirements satisfied

No transitive dependency

Function:



The screenshot shows the MySQL Workbench interface with a routine editor. The routine name is 'checknf3a'. The DDL (Data Definition Language) code is as follows:

```
1 • CREATE DEFINER=`root`@`localhost` FUNCTION `checknf3a`(a int ) RETURNS varchar(20) CHARSET utf8mb4
2     READS SQL DATA
3     DETERMINISTIC
4     BEGIN
5         declare result varchar(20);
6         SET GLOBAL log_bin_trust_function_creators = 1;
7         IF a>1 THEN
8             SET result = "IN NF 3 ";
9         ELSE
10            SET result = 'not IN NF 3 ';
11        END IF;
12        RETURN result;
13    END
```

The interface includes a toolbar with icons for file operations, a search bar, and a status bar at the bottom with 'Apply' and 'Revert' buttons. A 'Routine' tab is visible at the bottom left.

Procedure:

Name: `new_procedure2` The name of the routine is parsed automatically from the DDL statement. The DDL is parsed automatically while you type.

DDL:

```
1 • CREATE DEFINER='root'@'localhost' PROCEDURE `new_procedure2`(in val int ,out var int)
2   BEGIN
3   select count(distinct Brand) into var from dmdd.ndtv_data_final where Id =val;
4   END
```

Routine

Apply Revert

Limit to 2000 rows

```
1 • set @var = 0;
2 • call dmdd.new_query(6,@var);
3 • select @var;
4
5 • select dmdd.`check 2nf`(5);
6
7
```

Outputs:

