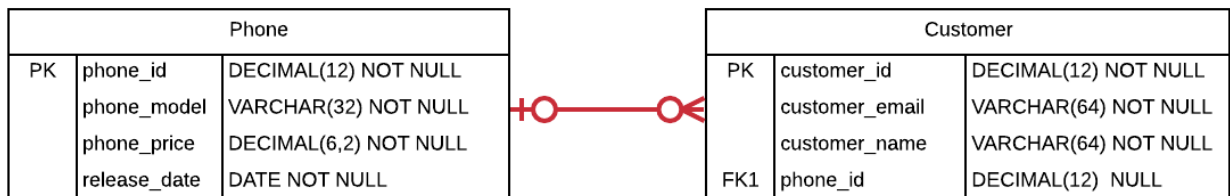


Section One – Relating Data

Section Background

To practice relating data, you will be working with the following simplified Phone schema:



In this schema, the Phone table contains a primary key, the name of the phone model (for example, “Apple iPhone X”), the date when the phone is released to the general public, and the price for that Phone. The Customer table contains a primary key, the name of the Customer, their email, and a foreign key that references the phone the customer is purchasing. The foreign key enforces the relationship between Phone and Customer so that a phone can be purchased by many customers, but a customer can only purchase one phone at a time. The foreign key is nullable since a customer may purchase a phone the store has in stock. There can also be phone that has not been purchased by any customer.

The schema is intentionally simplified when compared to what you might see in a real-world production schema. The schema does not record a history of Phone price changes as the cost changes, nor does it support special fee reductions during special periods. Many other attributes that would exist in a production database are not present. The current complexity is sufficient; additional complexity in the schema would not aid your learning at this point.

Do not worry if you don’t yet fully understand foreign keys and relationships. The Lab 2 explanations document gives you the information you need to complete the steps in this lab.

As a reminder, for each step that requires SQL, make sure to capture a screenshot of the command and the results of its execution.

Section Steps

1. **Creating the Table Structure** – Create the Phone and Customer tables, including all their columns, datatypes, and constraints, and the foreign key constraint.

```
1 CREATE TABLE Phone(  
2     phone_id DECIMAL(12) NOT NULL PRIMARY KEY,  
3     phone_model VARCHAR(32) NOT NULL,  
4     phone_price DECIMAL(6,2) NOT NULL,  
5     release_date DATE NOT NULL  
6 );  
7  
8
```









Data output Messages Notifications

CREATE TABLE

Query returned successfully in 177 msec.

```
1 CREATE TABLE Phone(  
2     phone_id DECIMAL(12) NOT NULL PRIMARY KEY,  
3     phone_model VARCHAR(32) NOT NULL,  
4     phone_price DECIMAL(6,2) NOT NULL,  
5     release_date DATE NOT NULL  
6 );  
7  
8 SELECT * from Phone;
```

Data output Messages Notifications

							
phone_id	phone_model	phone_price	release_date				
[PK] numeric (12)	character varying (32)	numeric (6,2)	date				

Query Query History

```
1 CREATE TABLE Customer(  
2     customer_id DECIMAL(12) NOT NULL PRIMARY KEY,  
3     customer_email VARCHAR(64) NOT NULL,  
4     customer_name VARCHAR(64) NOT NULL,  
5     phone_id DECIMAL(12) NULL,  
6     CONSTRAINT phone_id_fk  
7     FOREIGN KEY (phone_id)  
8     REFERENCES Phone(phone_id)  
9 );  
10  
11
```

Data output Messages Notifications

CREATE TABLE

Query returned successfully in 48 msec.

Query Query History

```
1 CREATE TABLE Customer(  
2     customer_id DECIMAL(12) NOT NULL PRIMARY KEY,  
3     customer_email VARCHAR(64) NOT NULL,  
4     customer_name VARCHAR(64) NOT NULL,  
5     phone_id DECIMAL(12) NULL,  
6     CONSTRAINT phone_id_fk  
7     FOREIGN KEY (phone_id)  
8     REFERENCES Phone(phone_id)  
9 );  
10  
11 SELECT * FROM Customer;
```

Data output Messages Notifications



customer_id [PK] numeric (12)	customer_email character varying (64)	customer_name character varying (64)	phone_id numeric (12)
----------------------------------	--	---	--------------------------

2. Populating the Tables – Insert the following rows into the Phone table.

Phone 1

name = Apple iPhone X
release_date = 11/03/2017
price = \$379

Phone 2

name = Galaxy S21+
release_date = 01/29/2021
price = \$799

Phone 3

name = Xenos 360
release_date = 03/22/2021
price = \$1,024

Phone 4

name = Meridian Duplex
release_date = 05/15/2021
price = \$462

Insert five customers of your choosing into the Customer table. Please note the following.
The first customer must not be associated with any phone because they have not yet purchased any phone, and the first phone (Apple iPhone X) must not be associated with any customer because no one has yet purchased it. The rest of the customers should be associated with phones, and the rest of the phones should be associated with customers.

Select all rows in both tables to view what you inserted.

Query Query History

```
1 INSERT INTO Phone (phone_id, phone_model, release_date, phone_price)
2 VALUES
3     (1, 'Apple iPhone X', CAST('11/03/2017' AS DATE), 379),
4     (2, 'Galaxy S21+', CAST('01/29/2021' AS DATE), 799),
5     (3, 'Xenos 360', CAST('03/22/2021' AS DATE), 1024),
6     (4, 'Meridian Duplex', CAST('05/15/2021' AS DATE), 462);
```

Data output Messages Notifications

INSERT 0 4

Query returned successfully in 164 msec.

Query Query History

```
1 INSERT INTO Phone (phone_id, phone_model, release_date, phone_price)
2 VALUES
3     (1, 'Apple iPhone X', CAST('11/03/2017' AS DATE), 379),
4     (2, 'Galaxy S21+', CAST('01/29/2021' AS DATE), 799),
5     (3, 'Xenos 360', CAST('03/22/2021' AS DATE), 1024),
6     (4, 'Meridian Duplex', CAST('05/15/2021' AS DATE), 462);
7
8 SELECT * FROM Phone;
```

Data output Messages Notifications



	phone_id [PK] numeric (12)	phone_model character varying (32)	phone_price numeric (6,2)	release_date date
1	1	Apple iPhone X	379.00	2017-11-03
2	2	Galaxy S21+	799.00	2021-01-29
3	3	Xenos 360	1024.00	2021-03-22
4	4	Meridian Duplex	462.00	2021-05-15

Query

Query History

1

INSERT INTO Customer(customer_id, customer_email,customer_name, phone_id)

2

VALUES

3

(10, 'mike@gmail.com', 'MIKE', NULL),

4

(11, 'gritzie@gmail.com', 'GRITZIE', 2),

5

(12, 'jsmart@gmail.com', 'John Smart', 3),

6

(13, 'braynt@gmail.com', 'Bryant Ryan', 4),

7

(14, 'lindsay@outlook.com', 'Lindsay Gambin', 2);

8

9

10

11

12

13

Data output

Messages

Notifications

INSERT 0 5

Query returned successfully in 4 min 56 secs.

Query

Query History

1

INSERT INTO Customer(customer_id, customer_email,customer_name, phone_id)

2

VALUES

3

(10, 'mike@gmail.com', 'MIKE', NULL),

4

(11, 'gritzie@gmail.com', 'GRITZIE', 2),

5

(12, 'jsmart@gmail.com', 'John Smart', 3),

6

(13, 'braynt@gmail.com', 'Bryant Ryan', 4),

7

(14, 'lindsay@outlook.com', 'Lindsay Gambin', 2);

8

9

SELECT * FROM Customer;

10

11

12

13

Data output

Messages

Notifications

+

📄

▼

📋

🗑️


🔍

⬇️

📈

	customer_id [PK] numeric (12)	customer_email character varying (64)	customer_name character varying (64)	phone_id numeric (12)
1	10	mike@gmail.com	MIKE	[null]
2	11	gritzie@gmail.com	GRITZIE	2
3	12	jsmart@gmail.com	John Smart	3
4	13	braynt@gmail.com	Bryant Ryan	4
5	14	lindsay@outlook.com	Lindsay Gambin	2

3. **Invalid Reference Attempt** – As an exercise, attempt to insert a Customer that references a Phone that doesn't exist. Summarize:



The screenshot shows a database query interface with two tabs: 'Query' and 'Query History'. The 'Query' tab is active, displaying an SQL insert statement. The statement is as follows:

```
1 INSERT INTO Customer(customer_id, customer_email,  
2 customer_name, phone_id)  
3 VALUES  
4 (15, 'sophia@gmail.com', 'Sophia Ng', 5);  
5  
6  
7  
8  
9  
10
```

Below the query, there are three tabs: 'Data output', 'Messages', and 'Notifications'. The 'Messages' tab is active, displaying an error message:

```
ERROR: insert or update on table "customer" violates foreign key constraint "phone_id_fk"  
DETAIL: Key (phone_id)=(5) is not present in table "phone".  
SQL state: 23503
```

a. **why the insertion failed, and**

The insertion failed because the database can't find the "phone_id" with 5 in the phone table. According to my example above, I tried to insert a row reference to the "phone_id = 5" which does not exist in the parent table (Phone table). This action causes the foreign key violation.

b. **how you would interpret the error message from your RDBMS so that you know that the error indicates the Phone reference is invalid.**

I am using Postgresql and the error message show in the above screenshot. The message shows very clearly that "phone_id" I tried to input in the table "Customer" violates foreign key constraint, named "phone_id_fk". Because I insert "phone_id = 5" which is not exist in the parent table "Phone", phone_id only has values such as 1, 2, 3, 4. I set the foreign key constraint is "phone_id_fk" I interpreted as the constraint defines a foreign key from the "Customer" table, the letter "fk" stand for "foreign key".

4. **Listing Matches** – With a single SQL query, fulfill the following request:

List the names of the Phones that have Customers, and the names of all the Customers that have a Phone.

Query

Query History

```

1 SELECT phone_model, customer_name
2 FROM Phone
3 JOIN Customer ON Phone.phone_id = Customer.phone_id;
4
5
6 |
7

```

Data output

Messages

Notifications

+

📄

▼

📋

🗑️

📦

⬇️

📈

	phone_model character varying (32) 🔒	customer_name character varying (64) 🔒
1	Galaxy S21+	GRITZIE
2	Xenos 360	John Smart
3	Meridian Duplex	Bryant Ryan
4	Galaxy S21+	Lindsay Gambin

- From a technical SQL perspective, explain why some rows in the Phone table and some rows in the Customers table were not listed.

This step is defined as inner join, which only list rows that match the join condition, all other rows which don't match the join condition are removed. In this step I only list the name of phones that have customers, and the name of all customers that have a phone. Firstly, I select 2 columns such as phone_model and customer_name from Phone table and then join to Customer table by the JOIN keyword. ON keyword indicates what Boolean expression will be used to determine which rows match. The condition in this step is Phone.phone_id = Customer.phone_id. The row "Apple iPhone X" from Phone table and "Mike" from Customers table were not listed because they don't match with the condition Phone.phone_id = Customer.phone_id.

5. Listing All from One Table – Fulfill the following request:

List the names and release date of all Phones whether or not they have been purchased by Customers. For the Phones that were purchased by customers Customers, list the names of the Customers that have those Phones. Order the list by the release date, oldest to newest.

There are two kinds of joins that can be used to satisfy this request. Write two queries using each type of join to satisfy this request.

Query
Query History

```

1 SELECT phone_model, release_date, customer_name
2 FROM Phone
3 LEFT JOIN Customer ON Phone.phone_id = Customer.phone_id
4 ORDER BY release_date ASC;
5
6
7

```

Data output
Messages
Notifications

+

📄

▼

📋

🗑️

📦

⬇️

📈

	phone_model character varying (32) 🔒	release_date date 🔒	customer_name character varying (64) 🔒
1	Apple iPhone X	2017-11-03	[null]
2	Galaxy S21+	2021-01-29	GRITZIE
3	Galaxy S21+	2021-01-29	Lindsay Gambin
4	Xenos 360	2021-03-22	John Smart
5	Meridian Duplex	2021-05-15	Bryant Ryan

Query
Query History

```

1 SELECT phone_model, release_date, customer_name
2 FROM Customer
3 RIGHT JOIN Phone ON Phone.phone_id = customer.phone_id
4 ORDER BY release_date;

```

Data output
Messages
Notifications

+

📄

▼

📋

🗑️

📦

⬇️

📈

	phone_model character varying (32) 🔒	release_date date 🔒	customer_name character varying (64) 🔒
1	Apple iPhone X	2017-11-03	[null]
2	Galaxy S21+	2021-01-29	GRITZIE
3	Galaxy S21+	2021-01-29	Lindsay Gambin
4	Xenos 360	2021-03-22	John Smart
5	Meridian Duplex	2021-05-15	Bryant Ryan

6. Listing All from Another Table – Fulfill the following request:

List the names of all Customers whether or not they have purchased a Phone, and the names of the Phones which Customers have purchased. Order the list by Customer email in reverse alphabetical order.

Just as with step #5, there are two kinds of joins that can be used to satisfy this request. Write two queries using each type of join to satisfy this request.

Query

Query History

1

2

3

4

5

6

7

SELECT

phone_model,customer_name

FROM

Phone

RIGHT JOIN

Customer

ON

Phone.phone_id = Customer.phone_id

ORDER BY

customer_email

DESC;

Data output

Messages

Notifications

phone_model

character varying (32)

customer_name

character varying (64)

1

[null]

MIKE

2

Galaxy S21+

Lindsay Gambin

3

Xenos 360

John Smart

4

Galaxy S21+

GRITZIE

5

Meridian Duplex

Bryant Ryan

Query

Query History

1

2

3

4

SELECT customer_name, phone_model
FROM Customer
LEFT JOIN Phone ON phone.phone_id = Customer.phone_id
ORDER BY customer_email DESC;

Data output

Messages

Notifications

≡

📄

▼

📋

🗑️

🗄️

⬇️

📈

	customer_name character varying (64) 🔒	phone_model character varying (32) 🔒
1	MIKE	[null]
2	Lindsay Gambin	Galaxy S21+
3	John Smart	Xenos 360
4	GRITZIE	Galaxy S21+
5	Bryant Ryan	Meridian Duplex

7. Listing All from Both Tables – Fulfill the following request with a single SQL query:

List the names of all Phones and the emails of all Customers, as well as which Phones have which Customers. Order the list alphabetically by Phone name then by Customer name.

Query Query History

```

1 SELECT Phone_model, Customer_email, Customer_name
2 FROM Phone
3 FULL JOIN Customer ON Customer.phone_id = Phone.phone_id
4 ORDER BY (Phone_model, Customer_name)

```

Data output Messages Notifications

	phone_model character varying (32)	customer_email character varying (64)	customer_name character varying (64)
1	Apple iPhone X	[null]	[null]
2	Galaxy S21+	gritzie@gmail.com	GRITZIE
3	Galaxy S21+	lindsay@outlook.com	Lindsay Gambin
4	Meridian Duplex	braynt@gmail.com	Bryant Ryan
5	Xenos 360	jsmart@gmail.com	John Smart
6	[null]	mike@gmail.com	MIKE

Section Two – Expressing Data

Section Background

While it is certainly useful to directly extract values as they are stored in a database, it is more useful in some contexts to manipulate these values to derive a different result. In this section we practice using value manipulation techniques to transform data values in useful ways. For example, what if we want to tell a customer exactly how much money they need to give for a purchase? We could extract a price and sales tax from the database, but it would be more useful to compute a price with tax as a single value by multiplying the two together and rounding appropriately, and formatting it as a currency, as illustrated in the figure below.

<i>Less Useful to Customer</i>		<i>More Useful to Customer</i>
price	tax_percent	price_with_tax
7.99	8.5	\$8.67

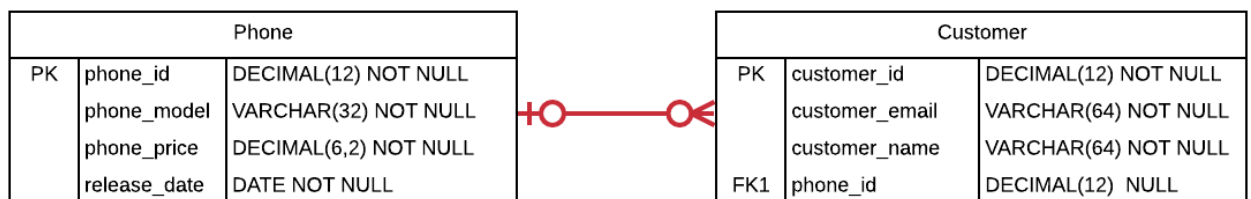
We do not need to store the price with tax, because we can derive it when we need it.

As another example, what if we need to send an email communication to a customer by name? We could extract the prefix, first name, and last name of the customer, but it would be more useful to properly format the name by combining them in proper order, as illustrated below.

<i>Less Useful to Customer</i>			<i>More Useful to Customer</i>
prefix	first_name	last_name	name
Mr.	Seth	Nemes	Mr. Seth Nemes

Again, we do not need to store the formatted name, because we can derive it when we need it from its constituent parts. Manipulating raw data values stored in database tables can yield a variety of useful results we need without adding the burden of storing every such result.

In this section, you use expressions to manipulate and format data values. The first several steps in this section teach you several important concepts needed to correctly use expressions, including attributes of SQL clients, operator precedence, datatype precedence, and formatting functions. The later steps have you use this knowledge to manipulate and format data values. You work with the same Phone and Customer schema from Section One. The schema is illustrated below again for your review.



Section Steps

8. *Formatting as Money* – Fulfill the following request with a single query:

The managers of the Phone store want to review their prices. List the names and prices of all Phones, making sure to format the price monetarily in U.S. dollars (for example, “\$379.00”).

Query

Query History

1

2

SELECT

Phone_model,

to_char(Phone_price,

'\$9999.99')

AS

Phone_price_USD

FROM

Phone;

Data output

Messages

Notifications

≡+

📄

▼

📋

🗑️

🗄️

⬇️

📈

	phone_model character varying (32) 🔒	phone_price_usd text 🔒
1	Apple iPhone X	\$ 379.00
2	Galaxy S21+	\$ 799.00
3	Xenos 360	\$ 1024.00
4	Meridian Duplex	\$ 462.00

9. Using Expressions – Fulfill the following request with a single query:

The managers of the Phone store are looking to increase purchases of Phones by lowering prices by \$50. List the names and discounted prices of all Phones, making sure to format the price monetarily in U.S. dollars.

Query

Query History

1

SELECT Phone_model, to_char(Phone_price - 50, '\$9999.99') as discount_price









2

FROM Phone;

Data output

Messages

Notifications



	phone_model character varying (32)	discount_price text
1	Apple iPhone X	\$ 329.00
2	Galaxy S21+	\$ 749.00
3	Xenos 360	\$ 974.00
4	Meridian Duplex	\$ 412.00

10. Advanced Formatting – Fulfill the following request with a single query:

The managers want to determine what Phone each Customer has purchased as well as its price, and wants the list ordered by Customer name. For example, if the Apple iPhone X has a price of \$379, and has two Customers – John Doe and Jane Doe – the results would have two lines for this Phone:

John Doe (Apple iPhone X - \$379.00)

Jane Doe (Apple iPhone X - \$379.00)

Query Query History

```
1 SELECT Customer_name || ' (' || phone_model || ' - ' || to_char(phone_price, '$9999.99') || ')' AS Advance_Formatting
2 FROM Customer
3 JOIN Phone ON Customer.phone_id = Phone.phone_id
4 ORDER BY Customer_name;
```

Data output Messages Notifications



	advance_formatting text
1	Bryant Ryan (Meridian Duplex - \$ 462.00)
2	GRITZIE (Galaxy S21+ - \$ 799.00)
3	John Smart (Xenos 360 - \$ 1024.00)
4	Lindsay Gambin (Galaxy S21+ - \$ 799.00)

Section Three – Advanced Data Expression

Section Background

Boolean expressions can become complex, yet are essential to filtering results in SQL. Boolean expressions can determine if a set of columns meet a possibly complex set of conditions. In this section, you learn to work with more advanced Boolean expressions.

Modern relational databases have the ability to calculate a column automatically. Such a column is identified by many terms – *generated*, *computed*, *calculated*, *derived*, and *virtual*. If one column can be calculated from the values in other columns, it is best practice to avoid storing the extra value, because it can become out of sync with the other columns. That is, if one of the columns change value, but the derived column is not updated, the data becomes inconsistent. In fact, storing derived columns is a form of data redundancy.

As described in Section Two, one option to avoid storage is to dynamically calculate derived values in a SQL query using an expression. Another option, the topic of this section, is to create a generated column, and have the database calculate it automatically.

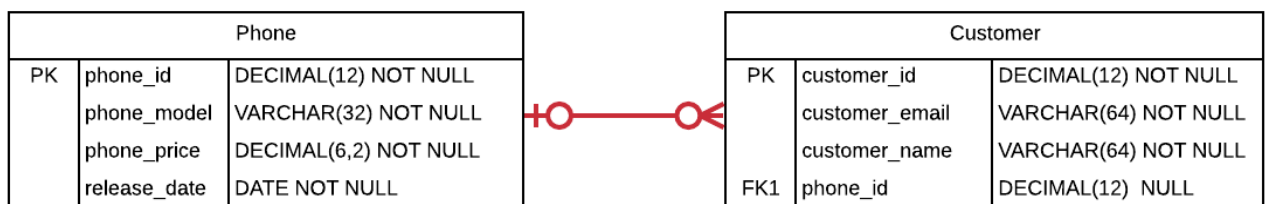
Using an example from Section Two, if one column contains a price, and another column contains a tax percentage, a third column could contain the price after tax.

price	tax_percent
7.99	8.5

price_with_tax
\$8.67

This third value can be calculated by multiplying the price by the tax percentage, and performing proper rounding to two decimal points. We would not want to store the price with tax, because we can derive it. In this section, you learn how to create generated columns to contain values that can be derived from other columns.

You work with the same Phone and Customers schema from prior sections. The schema is illustrated below again for your review.



Section Steps

11. Evaluating Boolean Expressions – Indicate the final values for each of the Boolean expressions below. You must show your work for full credit, by showing the value of each operation step-by-step.

a. $(\text{true AND false}) \text{ OR } (\text{false AND true}) = \text{false}$

Because we have:

- $\text{true AND false} = \text{false}$
- $\text{false AND true} = \text{false}$

→ *Reduction of inner expression:*

$(\text{true AND false}) \text{ OR } (\text{false AND true}) = (\text{false}) \text{ OR } (\text{false})$

We also have:

- $(\text{false}) \text{ OR } (\text{false}) = \text{false}$

The final value for this Boolean expression **$(\text{true AND false}) \text{ OR } (\text{false AND true})$** is **false**

b. $(\text{true OR true}) \text{ AND NOT}(\text{false OR true}) \text{ AND } (\text{true AND true}) = \text{false}$

Because we have:

- $\text{true OR true} = \text{true}$
- $\text{false OR true} = \text{true}$ → **We have:** $\text{NOT}(\text{false OR true}) = \text{NOT}(\text{true}) = \text{false}$
- $\text{true AND true} = \text{true}$

→ *Reduction of expression step#1:*

$(\text{true OR true}) \text{ AND NOT}(\text{false OR true}) \text{ AND } (\text{true AND true}) = (\text{true}) \text{ AND } (\text{false}) \text{ AND } (\text{true})$

We also have:

- $\text{true AND false} = \text{false}$

→ *Reduction of expression step#2:*

$(\text{true}) \text{ AND } (\text{false}) \text{ AND } (\text{true}) = (\text{false}) \text{ AND } (\text{true})$

Finally, we have:

- $(\text{false}) \text{ AND } (\text{true}) = \text{false}$

The final value for this Boolean expression

$(\text{true OR true}) \text{ AND NOT}(\text{false OR true}) \text{ AND } (\text{true AND true})$ is **false**

c. **NOT((false OR false) AND NOT(true AND true) AND (true OR false)) = true**

Because we have:

- false OR false = false
- true AND true = true → **We have:** NOT(true AND true) = NOT(true) = false
- true OR false = true

→ **Reduction inner expression step#1:**

NOT((false OR false) AND NOT(true AND true) AND (true OR false)) = NOT((false) AND (false) AND (true))

→ **Reduction of expression is:** NOT((false) AND (false) AND (true))

We also have:

- false AND false = false

→ **Reduction inner expression step#2:**

- NOT((false) AND (false) AND (true)) = NOT((false) AND (true))

We also have:

- false AND true = false

→ **Reduction inner expression step#3:**

- NOT((false) AND (true)) = NOT(false)

→ **Reduction expression step#4:**

- NOT(false) = true

The final value for this Boolean expression

NOT((false OR false) AND NOT(true AND true) AND (true OR false)) is true

12. Using Boolean Expressions in Queries – Address the following scenarios.

a. Any Phone matching the following condition is considered a high-end Phone for the store: Any Phone, except for the “Apple iPhone X” Phone, that is available on or after 05/01/2020, with a price of \$900.00 or higher, is a high-end Phone. Write a query that shows the name and price of all high-end Phones. It’s fine if you’d like to insert another row of Phones to become the high-end Phone.

Query

Query History

1

2

3

4

5

6

7

SELECT phone_model, phone_price
FROM Phone
WHERE NOT phone_model = 'Apple iPhone X'
AND release_date >= CAST('05/01/2020' AS DATE)
AND phone_price >= 900;

Data output

Messages

Notifications

≡+

📄

▼

📋

🗑️

🗄️

⬇️

📈

	phone_model character varying (32) 🔒	phone_price numeric (6,2) 🔒
1	Xenos 360	1024.00

b. The management company also has one *deluxe Phone* that sets it apart from other Phones. First, define your own conditions for this Phone, making sure the conditions include the Phone name, release date, and the phone price. Then write a query that shows the name and price of the Phone. It's fine if you'd like to insert another row of Phones to become the deluxe Phone.

- Deluxe phone is considered: Any phone, except for the "Xenos 360" phone, that is available on 29 – Jan – 2021, with price more than \$700.

Query

Query History

1

SELECT phone_model, phone_price

2

FROM Phone

3

WHERE

4

NOT phone_model = 'Xenos 360'

5

AND phone_price > 700

6

AND release_date = CAST('29-Jan-2021' AS DATE);

Data output

Messages

Notifications

≡+

📄

▼

📋

🗑️

🗄️

⬇️

📈

	phone_model character varying (32) 🔒	phone_price numeric (6,2) 🔒
1	Galaxy S21+	799.00

13. Using Generated Columns – Address the following.

a. Define a new generated column named *reduced_price*, which gives a lower price for the Phone for when the store wants to increase purchases by lowering prices (such as after the Christmas holiday shopping season). You determine the percentage or fixed value discount for these Phones. Then write a query that lists out the name of all Phones, along with their regular and reduced prices.

- I define fixed value discount for phones: \$100

Generate reduced price column:

Query

Query History

1

ALTER TABLE Phone

2

ADD reduced_price DECIMAL(12) GENERATED ALWAYS AS (phone_price - 100) STORED;

3

4

SELECT * FROM Phone;

Data output

Messages

Notifications

≡

📄

▼






📋

🗑️

📦

⬇️

📈

	phone_id [PK] numeric (12) 	phone_model character varying (32) 	phone_price numeric (6,2) 	release_date date 	reduced_price numeric (12) 
1	1	Apple iPhone X	379.00	2017-11-03	279
2	2	Galaxy S21+	799.00	2021-01-29	699
3	3	Xenos 360	1024.00	2021-03-22	924
4	4	Meridian Duplex	462.00	2021-05-15	362

List out all phone name, regular price and reduced price:

Query

Query History

1

SELECT phone_model, phone_price AS regular_price, reduced_price

2

FROM Phone;

Data output

Messages

Notifications

≡

📄

▼

📋

🗑️

📦

⬇️

📈

	phone_model character varying (32) 🔒	regular_price numeric (6,2) 🔒	reduced_price numeric (12) 🔒
1	Apple iPhone X	379.00	279
2	Galaxy S21+	799.00	699
3	Xenos 360	1024.00	924
4	Meridian Duplex	462.00	362

b. Address #12a again in a different way. First, define a generated column named *is_high_end* on the Phone table, which indicates whether it's a high-end Phone or not. Then write a query that lists only those Phones. Include relevant columns in the result.

Generate column named is_high_end:

Query Query History

1

ALTER TABLE Phone

2

ADD is_high_end BOOLEAN GENERATED ALWAYS AS

3

(CASE

4

WHEN NOT phone_model = 'Apple iPhoneX'

5

AND release_date >= CAST('05/01/2020' AS DATE)

6

AND phone_price >= 900 THEN true

7

ELSE false

8

END) STORED;

Data output Messages Notifications

ALTER TABLE

Query returned successfully in 87 msec.

Query Query History

1

ALTER TABLE Phone

2

ADD is_high_end BOOLEAN GENERATED ALWAYS AS

3

(CASE

4

WHEN NOT phone_model = 'Apple iPhoneX'

5

AND release_date >= CAST('05/01/2020' AS DATE)

6

AND phone_price >= 900 THEN true

7

ELSE false

8

END) STORED;

9

10

SELECT * FROM Phone;

Data output Messages Notifications

	phone_id [PK] numeric (12)	phone_model character varying (32)	phone_price numeric (6,2)	release_date date	reduced_price numeric (12)	is_high_end boolean
1	1	Apple iPhone X	379.00	2017-11-03	279	false
2	2	Galaxy S21+	799.00	2021-01-29	699	false
3	3	Xenos 360	1024.00	2021-03-22	924	true
4	4	Meridian Duplex	462.00	2021-05-15	362	false

Query

Query History

```

1  SELECT * FROM Phone
2  WHERE is_high_end = true;

```

Data output

Messages

Notifications

≡+

📄

▼

📋

🗑️

🗄️

⬇️

📈

	phone_id [PK] numeric (12)	phone_model character varying (32)	phone_price numeric (6,2)	release_date date	reduced_price numeric (12)	is_high_end boolean
1	3	Xenos 360	1024.00	2021-03-22	924	true

Evaluation

Your lab will be reviewed by your facilitator or instructor with the criteria outlined in the table below. Note that the grading process:

- involves the grader assigning an appropriate letter grade to each criterion.
- uses the following letter-to-number grade mapping – A+=100,A=96,A-=92,B+=88,B=85,B-=82,C+=88,C=85,C-=82,D=67,F=0.
- provides an overall grade for the submission based upon the grade and weight assigned to each criterion.
- allows the grader to apply additional deductions or adjustments as appropriate for the submission.
- applies equally to every student in the course.

5 points per day will be subtracted for late submissions. Submissions beyond 5 days late will not be accepted. Please contact your facilitator for any exceptions.

Criterion	A	B	C	D	F
Section 1: Quality (30%)	The results for all steps in Section 1 are complete and correct. Appropriate SQL constructs have been used for all steps, and supporting explanations are present and accurate. All screenshots in Section 1 are legible. The section is well organized. All supporting explanations are clear and understandable.	The results for most steps in Section 1 are complete and correct. The appropriate SQL constructs have been used for most steps, and supporting explanations are mostly present and accurate. Most screenshots in Section 1 are legible. The section is organized. Most supporting explanations are clear and understandable.	The results for some steps in Section 1 are complete and correct. Appropriate SQL constructs have been used for some steps, and some supporting explanations are present and accurate. Some screenshots in Section 1 are legible. Some supporting explanations are clear and understandable.	The results for most steps in Section 1 are incomplete or incorrect. Appropriate SQL constructs have not been used for most steps. The screenshots in Section 1 are mostly illegible or missing. Most supporting explanations are unclear or missing. The section is disorganized.	The results for virtually all steps in Section 1 are incomplete or incorrect. Appropriate SQL constructs have not been used. Virtually all screenshots in Section 1 are illegible or missing. Virtually all supporting explanations are unclear or missing. The section is disorganized.
Section 2: Quality (30%)	The results for all steps in Section 2 are complete and correct. Appropriate SQL constructs have been used for all steps, and supporting explanations are present and accurate. All screenshots in Section 2 are legible. The section is well organized. All supporting explanations are clear and understandable.	The results for most steps in Section 2 are complete and correct. The appropriate SQL constructs have been used for most steps, and supporting explanations are mostly present and accurate. Most screenshots in Section 2 are legible. The section is organized. Most supporting explanations are clear and understandable.	The results for some steps in Section 2 are complete and correct. Appropriate SQL constructs have been used for some steps, and some supporting explanations are present and accurate. Some screenshots in Section 2 are legible. Some supporting explanations are clear and understandable.	The results for most steps in Section 2 are incomplete or incorrect. Appropriate SQL constructs have not been used for most steps. The screenshots in Section 2 are mostly illegible or missing. Most supporting explanations are unclear or missing. The section is disorganized.	The results for virtually all steps in Section 2 are incomplete or incorrect. Appropriate SQL constructs have not been used. Virtually all screenshots in Section 2 are illegible or missing. Virtually all supporting explanations are unclear or missing. The section is disorganized.
Section 3: #11 Soundness (10%)	The results of all Boolean expressions are correct. Step-by-step work for each expression is present and correct.	The results of most Boolean expressions are correct. Step-by-step work for most expressions is present and correct.	The results of some Boolean expressions are correct. Step-by-step work for some expressions is present and somewhat correct.	The results of all or almost all expressions are incorrect. Step-by-step work for some expressions is minimally present and partially correct.	The answer is missing, or all results are incorrect. Work has not been shown, or is entirely incorrect.

Section 3: #12a Soundness (10%)	The rows are listed correctly, and the logic used in the query is entirely accurate. All non-matching rows are excluded.	The rows are mostly listed correctly, and the logic used in the query is mostly accurate. Most non-matching rows are excluded.	The rows are listed somewhat correctly, and the logic used in the query is somewhat accurate. Some non-matching rows are excluded.	The rows listed are mostly incorrect, and the logic used in the query is mostly inaccurate. Non-matching rows may be included.	The answer is missing, or the query's results are entirely incorrect.
Section 3: #12b Soundness (10%)	The condition includes the required columns, and is reasonable in complexity. The correct row has been listed, and the logic used in the query is entirely accurate. All non-matching rows are excluded.	The condition includes the required columns. The correct row has been listed, and the logic used in the query is mostly accurate. Most non-matching rows are excluded.	The condition includes most required columns. The correct row may or may not be listed, and the logic used in the query is partly accurate. Some non-matching rows are excluded.	The condition includes some required columns. The correct row may or may not be listed, and the logic used in the query is mostly inaccurate. Non-matching rows may or may not be excluded.	The answer is missing, or the query's results are entirely incorrect.
Section 3: #13 Soundness (10%)	Both generated columns give accurate values for all rows. The logic for both is entirely accurate for current and any future rows. Only meaningful columns are included.	Both generated columns give accurate values for most rows. The logic for both is mostly accurate for current and any future rows. Mostly meaningful columns are included.	Both generated columns give accurate values for some rows. The logic for both is somewhat accurate for current and any future rows. Some meaningful columns are included.	Both generated columns give mostly inaccurate values. The logic for both is mostly inaccurate for current and any rows pizzas. Meaningful columns may not be included.	The answer is missing, or both generated columns give entirely inaccurate results. The logic for both columns is entirely unsound. Meaningful columns may not be included.

Use the **Ask the Teaching Team Forum** if you have any questions regarding how to approach this lab. Make sure to include your name in the filename and submit it in the *Assignments* section of the Ticket.