|  |
| --- |
| **Overview of the Lab** |
| One purpose of this lab it to teach you how to meaningfully relate data and to answer questions using related data, using SQL. In the prior lab you learned the fundamentals creating and using tables to store data, which is a good introduction to SQL. To be effective, however, you need to know how to work with relationships that naturally occur in the data you work with.  We don’t need to look far to recognize examples of relationships. For just some examples, people have addresses, pets are owned by owners, products are sold by stores, and cases are heard in a court. Relationships are both plentiful and inevitable in virtually any database. In many ways, the richness and complexity of the relationships in a database determines its usefulness in answering important questions about the data.  It is likewise inevitable with virtually any database data items need be formatted into human readable form, or manipulated to derive different results. Directly extracting values exactly as they are stored in a database works for some but not all queries. For example, a customer table may store a first and a last name, such as “Smith” and “Bob”, but emails or letters to the customer would use the full name, “Bob Smith”. Another objective of this lab is for you to learn how to format and manipulate data using functions and expressions.  From a technical perspective, together, we will learn:   * how to enforce relationships between two tables using a FOREIGN KEY constraint. * how to add related data to related tables. * how to ask questions and answer them using SQL queries that relate data. * details about the significant components that determine how a value is displayed in your SQL client. * to understand and effectively use expressions to manipulate data values. * how to use formatting functions to format data values into to human readable or other formats. * how to create and use more advanced Boolean expressions to solve more complex use cases. * how to create and use calculated columns. |

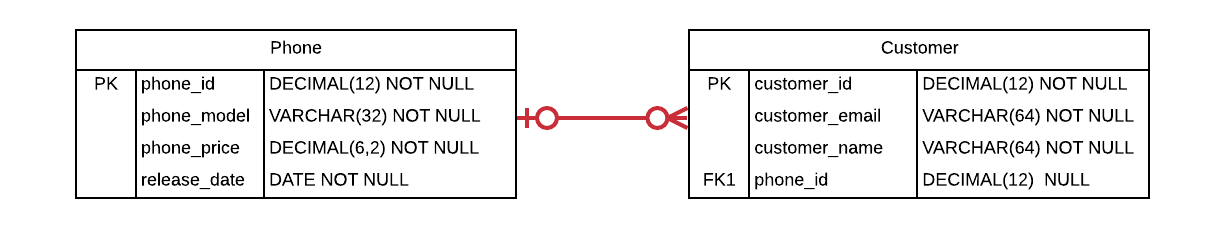
|  |
| --- |
| **Lab 2 Explanations Reminder** |
| As a reminder, it is important to read through the Lab 2 Explanation document to successfully complete this lab, available in the assignment inbox alongside this lab. The explanation document illustrates how to correctly execute each SQL construct step-by-step, and explains important theoretical and practical details. |

|  |
| --- |
| **Other Reminders** |
| * The examples in this lab will execute in modern versions of Oracle, Microsoft SQL Server, and PostgreSQL as is. * The screenshots in this lab display execution of SQL in the default SQL clients supported in the Ticket – Oracle SQL Developer, SQL Server Management Studio, and pgAdmin – but your screenshots may vary somewhat as different version of these clients are released. * Don’t forget to commit your changes if you work on the lab in different sittings, using the “COMMIT” command, so that you do not lose your work. |

**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.

With phone table: -

Graphical user interface, application

Description automatically generated

With customer table: -

Graphical user interface, text, application

Description automatically generated

1. *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.

Phone table after inserting values.

Table

Description automatically generated with medium confidence

customer table after populating values.

Table

Description automatically generated

1. *Invalid Reference Attempt –* As an exercise, attempt to insert a customer that references a Phone that doesn’t exist. Summarize:
   1. why the insertion failed, and

Answer: -

Graphical user interface, text, application, email

Description automatically generated

As we enforced the constraints, each row on table customer either must reference to one of the rows on phone table or must have value zero. In all other case, it won’t be able to references to phone table that is why it throws error. In our case, I am trying to reference row number 6 of customer table to row number 8 of the phone table, but there is no row number 8 on phone table that is why it threw error.

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

Answer: - While creating phone and customer table we made a rule to be followed by each entry into the customer table. As error says it violates foreign key constraint named ‘customer\_phone\_fk’ and further says ‘Key (phone\_id)=(8) is not present in table "phone"’. What is that means is the row we are trying to insert into the customer table cannot referenced to row number 8 of phone table because row 8(phone\_id=8) is not existed in phone table.

1. *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.  
     
   From a technical SQL perspective, explain why some rows in the Phone table and some rows in the Customers table were not listed.

List of phones that have Customers are: -

Graphical user interface, text, application

Description automatically generated

And list of customers that have phone: -

Graphical user interface, application

Description automatically generated

Full table customer with table looks like this where we can see list of customers that have phones and list of phones that corresponds customers.

Graphical user interface, application, Word

Description automatically generated

While inserting value into the table, we made a rule that first phones from phone table won’t have customer and first customer from customer table won’t have phone. That is why some row on phone table and some row on customer table were not listed. Also, in real world some phones are in stock and not purchased yet that is why all phones are not assigned to customers and vice versa.

1. *Listing All from One Table –* Fulfill the following request:  
     
   List the names and release date of all Phones whether they have been purchased by Customers. For the Phones that were purchased by 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.

Graphical user interface, application

Description automatically generated

Second kind of join with same result: -

Graphical user interface, application, Teams

Description automatically generated

1. *Listing All from Another Table –* Fulfill the following request:  
     
   List the names of all Customers whether 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.

Graphical user interface, application, Word

Description automatically generated

1. *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.

Graphical user interface, table

Description automatically generated

**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.

|  |  |
| --- | --- |
| *Less Useful to Customer* | |
| **price** | **tax\_percent** |
| 7.99 | 8.5 |

|  |
| --- |
| *More Useful to Customer* |
| **price\_with\_tax** |
| $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.

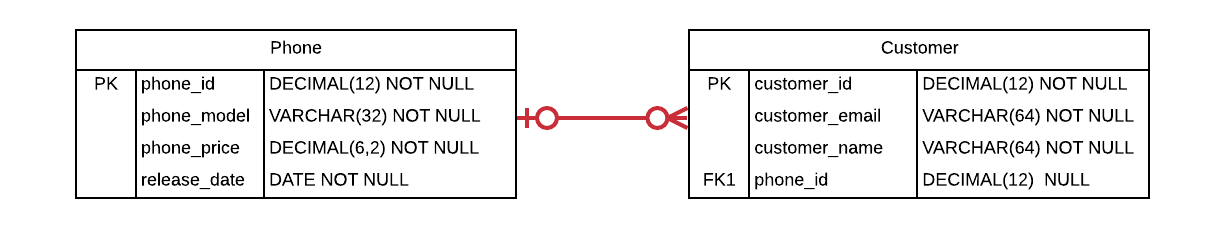
|  |  |  |
| --- | --- | --- |
| *Less Useful to Customer* | | |
| **prefix** | **first\_name** | **last\_name** |
| Mr. | Seth | Nemes |

|  |
| --- |
| *More Useful to Customer* |
| **name** |
| 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**

1. *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”).

Graphical user interface, application, Word, Teams

Description automatically generated

1. *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.

Answer: list of phone with reduced prices are given below.

Graphical user interface, application, Word

Description automatically generated

1. *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)

Answer: -

Graphical user interface, application, Word

Description automatically generated

**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.

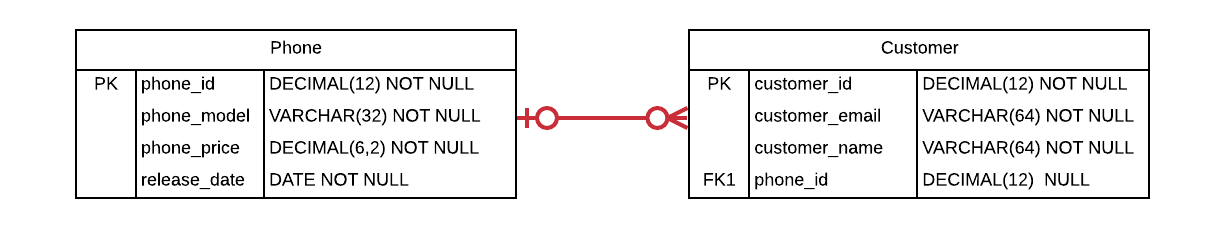
Using an example from Section Two, if one column contains a price, and another column contains a tax percentage, a third column could contain the price after tax.

|  |  |
| --- | --- |
|  | |
| **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**

1. *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.

Answer: - for AND operation, result of the operation is true if both operands are true otherwise result will be false.

* true AND true = true
* true AND false = false
* false AND true = false
* false and false = false

while in case of OR operation, result of operation will be false if both operands are false otherwise result will be true.

* true OR true = true
* true OR false = true
* false OR true = true
* false OR false = false
  1. (true AND false) OR (false AND true)

From above expression,

-true AND false = false

-false AND true = false

That is why,

-false or false =false

- Therefore (true AND false) OR (false AND true) = false

* 1. (true OR true) AND NOT(false OR true) AND (true AND true)

From above expression,

Simplifying expression inside parentheses first,

-true OR true = true

-false OR true = true

-true AND true = true

Now, above expression will be

=(true) AND NOT (true) AND (true)

=true AND false AND true # because NOT (true) ->false

=false AND true # because true AND false -> false

=false ->final answer

* 1. NOT ((false OR false) AND NOT(true AND true) AND (true OR false))

Simplifying expression inside parentheses first,

-false OR false -> false

-true AND true ->true

-true OR false - > true

Now above expression will be,

NOT((false) AND NOT (true) AND (true))

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

=NOT (false AND false AND true)

=NOT (false AND true) # combining underlined operands first

=NOT (false)

=true -> final answer

1. *Using Boolean Expressions in Queries* – Address the following scenarios.
   1. 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.

Answer: -

Graphical user interface, application, Teams

Description automatically generated

* 1. 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.

**For a deluxe phone, following criteria must be met:-**

price is between $750 to $1000

release date is on or after 01-Jul-2020

phone model is either GalaxyS21+ or Xenos 360.

Answer: -

Graphical user interface, application, Teams

Description automatically generated

1. *Using Generated Columns –* Address the following.
   1. 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.

**Answer: -**

**Criteria:**

store will be selling all their phone prices greater than $400 in reduce price. Reduced price will be $75 less than its original price.

write a query that lists out the name of all Phones, along with their regular and reduced prices.

Graphical user interface, application

Description automatically generated

* 1. 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.

Answer: -

**Criteria of high-end phone are:-**

-Price must be greater than $ 700

-Release date on or later than 05/19/2020

Graphical user interface, text, application, email

Description automatically generated

Altered table: -

Table

Description automatically generated

List of high-end phones: -

Graphical user interface, application

Description automatically generated

Extras: -

1.List of not High-end phones: -

Graphical user interface, application, table

Description automatically generated

**2.**

Instead of Boolean value we can also use text to categorize the data based on their value by using union without altering table as below

Graphical user interface, application

Description automatically generated

# 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.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Criterion** | **What it Measures** | **A+ Excellent** | **B Good** | **C Fair/Satisfactory** | **D Insufficient** | **F Failure** |
| **Section 1: Quality (30%)** | For section 1, this measures the correctness of the SQL results, the appropriateness of the SQL constructs used, how well the section is presented, and the quality of the supporting explanations. | Entirely correct results All constructs appropriate Excellent presentation Excellent supporting explanations | Mostly correct results Most constructs appropriate Good presentation Good supporting explanations | Somewhat correct results Some constructs appropriate Satisfactory presentation Satisfactory supporting explanations | Mostly incorrect results Inappropriate constructs Insufficient presentation Insufficient supporting explanations | Entirely incorrect results Inappropriate constructs Insufficient presentation Insufficient or missing supporting explanations |
| **Section 2: Quality (30%)** | For section 2, this measures the correctness of the SQL results, the appropriateness of the SQL constructs used, how well the section is presented, and the quality of the supporting explanations. | Entirely correct results All constructs appropriate Excellent presentation Excellent supporting explanations | Mostly correct results Most constructs appropriate Good presentation Good supporting explanations | Somewhat correct results Some constructs appropriate Satisfactory presentation Satisfactory supporting explanations | Mostly incorrect results Inappropriate constructs Insufficient presentation Insufficient supporting explanations | Entirely incorrect results Inappropriate constructs Insufficient presentation Insufficient or missing supporting explanations |
| **Section 3: #11 Soundness (10%)** | For #11, this measures the correctness of the results and the accuracy of the step-by-step work. | Results entirely correct Work entirely accurate | Results mostly correct Work mostly accurate | Results somewhat correct Work somewhat accurate | Results mostly incorrect Work mostly inaccurate | Results missing or entirely incorrect Work missing or entirely inaccurate |
| **Section 3: #12a Soundness (10%)** | For #12a, this measures the correctness of the rows listed and the soundness of the logic used in the query. | Rows entirely correct  All non-matching rows excluded Logic entirely sound | Rows mostly correct  Most non-matching rows excluded Logic mostly sound | Rows somewhat correct  Some non-matching rows excluded Logic somewhat sound | Rows mostly incorrect  Non-matching rows may be included Logic mostly unsound | Rows missing or entirely incorrect  Logic entirely unsound |
| **Section 3: #12b Soundness (10%)** | For #12b, this measures the soundness of the condition, the accuracy of the logic used in the query, and the correctness of the results. | Condition includes required columns Condition has reasonable complexity Logic entirely accurate Results entirely correct | Condition includes most required columns Condition has some complexity Logic mostly accurate Results mostly correct | Condition includes some required columns Logic somewhat accurate Results somewhat correct | Condition includes some required columns Logic mostly inaccurate Results mostly incorrect | Condition missing or excludes all required columns Logic missing or entirely inaccurate Results missing or entirely incorrect |
| **Section 3: #13 Soundness (10%)** | For #13, this measures the accuracy of the values for the generated columns, the accuracy of the logic, and the applicability of the columns included in the results. | Values for generated columns entirely accurate  Logic is entirely accurate  Only meaningful columns included | Values for generated columns mostly accurate  Logic is mostly accurate  Meaningful columns included | Values for generated columns somewhat accurate  Logic is somewhat accurate  Some meaningful columns included | Values for generated columns mostly inaccurate  Logic is mostly inaccurate  Few meaningful columns | Values for generated columns missing or entirely inaccurate  Logic missing or entirely inaccurate  Few meaningful columns |
|  |  | **Preliminary Grade:** |  | **Lateness Deduction** 5points per day4 days maximumContact your facilitator for any exceptions |  | **Lab Grade:** |

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.