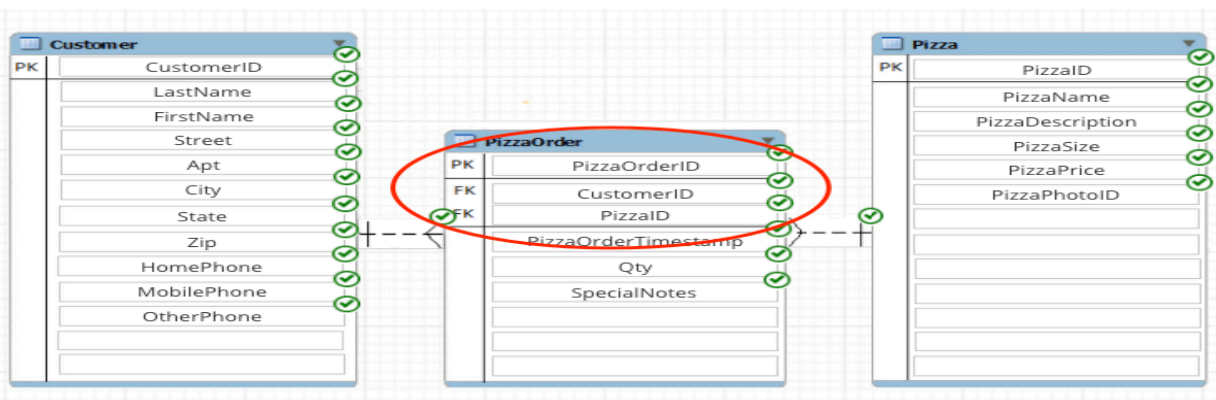



- Primary and foreign key are the same called Identifying relationship. (Solid line)



-Individual foreign key is called non-identifying relationship. (Dash line)


Creating a Database

You need to create a database named **PizzaOrderingDB**. Fill in the blank to complete the query in order to accomplish the task.

create database  PizzaOrderingDB;

Creating a Database

Create a database named CustomerDB. Fill in the blank to complete the query in order to accomplish the task.

create database  CustomerDB;

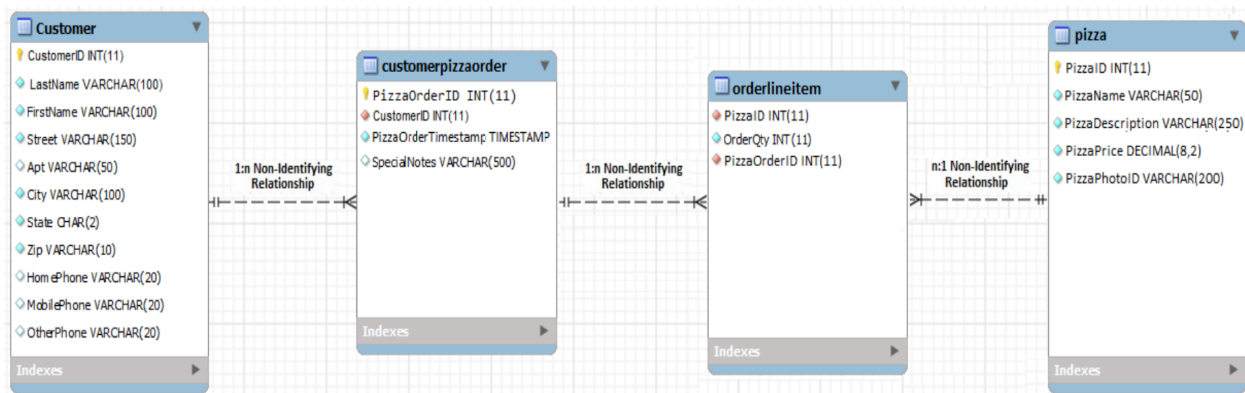
Dropping a Database

You need to delete a database named **PizzaOrderingDB**. Fill in the blank by dragging the correct command from the bottom to complete the query.

DROP DATABASE PizzaOrderingDB;

Create a table named **customerpizzaorder** represented in the following Physical Schema by using the **CREATE TABLE** command. **PizzaOrderID** must have **AUTO_INCREMENT** as the default value.

Hint: Use PizzaOrderTimestamp TIMESTAMP DEFAULT NOW() for setting the PizzaOrderTimestamp as the current date time. Table names are case sensitive. Primary and foreign keys cannot be null.



```
CREATE TABLE customerpizzaorder (
  PizzaOrderID INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
  CustomerID INT(11) NOT NULL,
  PizzaOrderTimestamp TIMESTAMP DEFAULT NOW(),
  SpecialNotes VARCHAR(500) ,
  FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID));
```

Use the **DESCRIBE** command to display the following table structure:

```
Describe Customer;
Describe Pizza;
Describe CustomerPizzaOrder;
```

Create a table of foods for a diet called **diet_plan**. The columns are:

- food_name (which should be a VARCHAR(20))
- calories (as INT)
- food_group (1 = dairy group, 2 = meat group, 3 = fruits and vegetables, 4 = bread group, 5 = misc, which is an INT)

```
CREATE TABLE diet_plan (
  food_name VARCHAR(20),
  calories INT,
  food_group INT
);
```

Create a table named **Cipher_keys** to store encryption keys for the data center model. The table should have the following data:

- Automatically generated integer primary key (**id**)
- The name of the key up to a length of 32 (required, **name**)
- Multi-line text field for the key (**cipher_key**)

```
CREATE TABLE Cipher_keys (id int auto_increment primary key, name varchar(32) not null, cipher_key text);
```

Create Tables in MySQL

Question 10 :

Create a table named uc_donutcustomers represented in the following Physical Schema by using the CREATE TABLE command.

Set the CustomerAddress column to required and the CustomerPhone default value to 555-555-5555.

uc_donutcustomers	
CustomerID	INT(11) +
CustomerAddress	TEXT
CustomerPhone	VARCHAR(30)

```
CREATE TABLE uc_donutcustomers (  
CustomerID INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,  
CustomerAddress TEXT NOT NULL,  
CustomerPhone VARCHAR(30) DEFAULT "555-555-5555");
```

Use the DESCRIBE command to display the following table structures:

```
Describe Customer;  
Describe Donut;  
Describe CustomerDonutOrder;
```

Creating a Table

Question 12 :

Create a table of sales for a diamond retail company called **diamond_sales**. The columns are:

- buyer_name (as VARCHAR(100))
- sales_price (as DECIMAL(8,2))
- cut_quality (1 = fair, 2 = good, 3 = very good, 4 = premium, 5 = ideal)
- purchase_date (as DATE)

```
CREATE TABLE diamond_sales (  
buyer_name VARCHAR(100),  
sales_price DECIMAL(8,2),  
cut_quality INT,  
purchase_date DATE  
);
```

Creating a Table to Store Login Information

Question 13 :

Create a table named **log_ins** to store user login information for a website. The table should have the following data:


- An automatically generated integer primary key (**id**)

- A multi-line text field for the IP address of the user (required, **ip_address**)
- A timestamp of when the user logged in (**log_time**)

```
CREATE TABLE log_ins (
id INT AUTO_INCREMENT PRIMARY KEY,
ip_address TEXT NOT NULL,
log_time TIMESTAMP);
```

Dropping a Database

Delete a database named CustomerDB. Fill in the blank by dragging the correct command from the bottom to complete the query.

 **DROP** DATABASE CustomerDB;

Update Data in a Table in MySQL

You need to update the **OrderQty** as **10** in the **uc_pizzaorder** table where **PizzaOrderID = 1**.

```
UPDATE uc_pizzaorder
SET OrderQty = 10
WHERE PizzaOrderID = 1;
```

Delete Rows in MySQL

A table, **test1** has already been created. Delete the rows where **id** is equal to **35** or **23**.

Structure of **test1** table is given below:

Field	Type	Null	Key	Default	Extra
name	varchar(30)	YES		NULL	
id	int(11)	NO	PRI	NULL	

Table A: test1

```
DELETE FROM test1
WHERE id = 23 OR id = 35;
```

Adding New Data in a Table

Consider the following **garage** table structure:

Field	Type	NULL
make	varchar(10)	YES
model	varchar(10)	YES
year	int(11)	YES

Table A: Garage Table Structure

Enter a new car in the **garage** table, a 1971 Chevrolet Camaro.

```
INSERT INTO garage VALUES ('Chevrolet', 'Camaro', 1971);
```

Deleting Specific Data from a Table

Consider the following **garage** table:

make	model	year
Ford	Fairlane	1961
Ferrari	250 GTO	1962
Ferrari	250 GTO	1963
Alfa Romeo	33 Stradale	1967
Alfa Romeo	33 Stradale	1970
Jaguar	XJ13	1965
Ferrari	330 P4	1967
Jaguar	E-Type	1961
Jaguar	E-Type	1964
Jaguar	E-Type	1967
Porsche	550	1953
Porsche	550	1954
Chevrolet	Corvette	1964
Chevrolet	Corvette	1967
Lamborghini	Miura	1967
Ferrari	Dino	1968

Mercedes-Benz	300SL	1955
Chevrolet	Camaro	1966
Chevrolet	Camaro	1969

Delete all the Ferraris from the **garage** table.

```
DELETE FROM garage WHERE make = 'Ferrari';
```

Copying Data from One Table to Another Table

Consider the following **garage** table:

The **for_sale** table structure is same as the **garage** table. You have to copy all of the "Alfa Romeo" cars from the **garage** table to the **for_sale** table.

```
INSERT INTO for_sale(make, model, year) (SELECT * FROM garage WHERE (make = 'Alfa Romeo'));
```

Updating Data in a Table

Change the price of all the cars to a more respectable \$50,000 in the **for_sale** table.

```
UPDATE for_sale SET price = 50000;
```

Updating Specific Data in a Table

Change the price for all the Camaro model to \$40,000 in the **for_sale** table.

```
UPDATE for_sale SET price = 40000 WHERE model = 'Camaro';
```

Adding Data in the Table

Consider the following table structure of **widgets**:

Field	Type	NULL	Key
num_sold	int(11)	YES	NULL
customer	varchar(40)	YES	NULL
paid_amt	double(10,2)	YES	NULL

Insert an order for "Mary Wills" for 3 widgets at a total of \$33.33.

```
INSERT INTO widgets VALUES (3, 'Mary Wills', 33.33);
```

Update Data in a Table in MySQL

Update **TakeOut** as Y in the **MenuItems** table where the **Cost** is more than 9.95.

```
UPDATE MenuItems  
SET TakeOut = "Y"  
WHERE Cost > 9.95;
```

Delete Rows in MySQL

Question 24 :

Delete the rows in the **MenuItems** table where **Popularity** is less than 3 and **TakeOut** is equal to Y.

```
DELETE FROM MenuItems  
WHERE Popularity < 3 AND TakeOut = "Y";
```

Adding New Data in a Table

Enter a new type 3 restaurant in the **Restaurants** table with the name Sarah's Salad Shack and the description, Fresh salads made to order.

```
INSERT INTO Restaurants  
(Name, Description, RestaurantType) VALUES ("Sarah's Salad Shack", "Fresh salads made to order", 3);
```

Deleting Specific Data from a Table

Question 26 :

Consider the following **garage** table:

```
DELETE FROM garage  
WHERE year < 1960;
```


Copying Data from One Table to Another Table

Question 27 :

Consider the following **uc_pizza** table:

PizzaID	PizzaName	PizzaDescription	PizzaPrice	PizzaPhotoID
1	Plain	Plain Pizza	1.5	www.Pizzaphotos.com/Pizza1.jpg
2	Pepperoni	Pepperoni Pizza	2	www.Pizzaphotos.com/Pizza2.jpg

The **Pizza** table structure is the same as the **uc_pizza** table. Copy the pizza with PizzaID = 1 from the **uc_pizza** table to the **Pizza** table.

```
INSERT INTO Pizza(PizzaID, PizzaName, PizzaDescription, PizzaPrice, PizzaPhotoID)
(SELECT * FROM uc_pizza WHERE PizzaID = 1);
```

Updating Data in a Table

Question 28 :

Change all menu items **DateAdded** dates to 2019-08-10 in the **MenuItems** table.

```
Update MenuItems
SET DateAdded = "2019-08-10";
```

Updating Specific Data in a Table

Question 29 :

Change all notes to be "large animal" and LastWeighDate to be 2019-08-10 in the **Animals** table where the Weight is more than 100 and Gender is M.

```
UPDATE Animals
SET Notes = "large animal", LastWeighDate = "2019-08-10"
WHERE Weight > 100 AND Gender = "M";
```

Adding Data in the Table

Question 30 :

Consider the following table structure of **Albums**:

Insert an album by Artist 3 you purchased on August 10, 2019 named "Reality All Stars" that costs \$17.50.

```
INSERT INTO Albums (Artist, Name, Cost, Purchased) VALUES  
(3, "Reality All Stars", 17.50, "2019-08-10");
```

Selecting Specific Columns from a Table

Consider the following **Buildings** table:

id	building_name	address	monthly_rent
101	Main office	6640 Main Street, Hillsbury, MA 01001	7300
102	Storage facility	6645 Main Street, Hillsbury, MA 01001	450

Table A: Buildings

Select just the building_name and monthly_rent from the **Buildings** table.

```
SELECT building_name, monthly_rent FROM Buildings;
```

Retrieving Data from a Table

Consider the following **garage** table:

Select all the "Ferrari" cars from the **garage** table.

```
SELECT * FROM garage WHERE make = 'Ferrari';
```

Retrieving Specific Column Data from a Table

Consider the following **Services** table:

id	company	monthly_fee
1001	McDougal Landscaping	500
1002	Ajax Cleaning Service	620
1003	Constance & Phillip's Catering	250
1004	Speedy Delivery Vehicles	170

Table A: Services

Select the company and monthly_fee from the **Services** table.

```
SELECT company, monthly_fee FROM Services;
```

Selecting Specific Columns from a Table

Consider the following **Animals** table:

Select just the Name, Weight and Gender from the **Animals** table.

```
SELECT Name, Weight, Gender FROM Animals;
```

Retrieving Data from a Table

Question 35 :

Consider the following **Artists** table:

Select all rows with a 2 rating from the **Artists** table.

```
SELECT * FROM Artists
WHERE Rating = 2;
```

Retrieving Specific Column Data from a Table

Question 36 :

Consider the following **Animals** table:

Select the name of all animals that are of **Species 2**.

```
SELECT Name FROM Animals
WHERE Species = 2;
```

Deleting a Table

Delete the **daily_diet_plan** table completely.

```
DROP TABLE daily_diet_plan;
```

Adding a Column in a Table

Add a new column called "day_of_the_week" to the **exercise_routine** table. The column type should be a char to store "M", "T" or "F".

```
ALTER TABLE exercise_routine ADD day_of_the_week char;
```

Modifying a Column in a Table

Run two DDL commands one after the other. The first one should delete the "exercise_name" column of the **exercise_routine** table. The second DDL command should create the "exercise_name" column again, but this time with the correct VARCHAR(45) data type.

```
ALTER TABLE exercise_routine DROP exercise_name;  
ALTER TABLE exercise_routine ADD exercise_name VARCHAR(45);
```

Adding a Comment

Add a comment to the **exercise_routine** table that says 'Exercise routine for January 1 to January 20'.

```
ALTER TABLE exercise_routine COMMENT 'Exercise routine for January 1 to January 20';
```

Deleting All the Data from a Table

Delete all the data from the **exercise_routine** table but leave the structure intact.

```
TRUNCATE TABLE exercise_routine;
```

Inserting Data from One Table to Another

Consider the following table structures:

Column Name	Data Type
id	int INT AUTO_INCREMENT PRIMARY KEY
color	VARCHAR(20)

Column Name	Data Type
object	VARCHAR(20)
color	VARCHAR(20)

Table B: Things Table Structure

Take distinct colors from the **Things** table and insert them into the **ColorLookup** table.

```
INSERT INTO ColorLookup (color) SELECT DISTINCT color FROM Things;
```

Adding a New Column in a Table

Add a new column to the **Things** table. It should be called "color_int" and be an integer.

```
ALTER TABLE Things ADD color_int INT;
```

Deleting a Column from a Table

Delete the "color" column from the **Things** table because this column is no longer needed.

```
ALTER TABLE Things DROP color;
```

Using ALTER Command to Modify a Column in a Table

You have to change the data type of "pass_key" to blob data type (this column should not be NULL) in the **Pass_keys** table.

```
ALTER TABLE Pass_keys modify pass_key blob NOT NULL;
```

Adding a Column in a Table to Store timestamp

You have to add the "created" column in the **Pass_keys** table that automatically holds the timestamp of when the row was created.

```
ALTER TABLE Pass_keys add created timestamp default current_time stamp;
```

Modifying a Column in a Table to Include a Comment

You have to change the "pass_key" column of blob data type to include the comment "id of employees" in the **Pass_keys** table.

```
ALTER TABLE Pass_keys MODIFY pass_key blob COMMENT 'id of employees';
```

Deleting a Table

Question 48 :

Delete the **MEATS** table completely.

```
DROP TABLE MEATS;
```

Adding a Column in a Table

Question 49 :

Add a new column called "P_budget" to the **project** table. The column type should be a decimal with a total of 8 digits to store the budget amount in US dollars with the appropriate placeholders for cents.

```
ALTER TABLE project ADD P_budget DECIMAL(8,2);
```

Modifying a Column in a Table

Question 50 :

Run two DDL commands one after the other. The first one should delete the "P_budget" column of the **UC_project** table. The second DDL command should create a "P_cost" column with the data type INT that is required.

```
ALTER TABLE UC_project DROP P_budget;  
ALTER TABLE UC_project ADD P_cost INT NOT NULL;
```

Adding a Comment

Question 51 :

Add a comment to the **Rating** table that says "Ratings of all movies released in 2019".

```
ALTER TABLE Rating COMMENT 'Ratings of all movies released in 2019';
```

Deleting All the Data from a Table

Question 52 :

Delete all the data from the **funds** table but leave the structure intact.

```
TRUNCATE TABLE funds;
```

Inserting Data from One Table to Another

Question 53 :

Consider the following table structures:

Take the names and ids of names beginning with the letter m through z from the **test1** table and insert them into the **test2** table.

```
INSERT INTO test2 (name,id) SELECT name,id FROM test1 WHERE name > "m";
```

Adding a New Column in a Table

Question 54 :

Add a new column to the **CHEESES** table. It should be called "description" and be a multi-lined text field that is required.

```
ALTER TABLE CHEESES ADD description TEXT NOT NULL;
```

Deleting a Column from a Table

Question 55 :

Delete the "description" column from the **UC_CHEESES** table because this column is no longer needed.

```
ALTER TABLE UC_CHEESES DROP description;
```

Using ALTER Command to Modify a Column in a Table

Question 56 :

Change the data type of "Description" to a variable length string between 1 and 50 characters in length in the **Restaurants** table. This column should not be NULL.

```
ALTER TABLE Restaurants modify Description VARCHAR(50) NOT NULL;
```

Adding a Column in a Table to Store Timestamp

Question 57 :

Add the "date_added" column in the **Places** table that automatically holds the timestamp of when the row was created.

```
ALTER TABLE Places ADD date_added timestamp default current_time stamp;
```

Modifying a Column in a Table to Include a Comment

Question 58 :

Change the popularity column of INT(11) data type to include the comment "Popularity based on 2019 sales" in the **MenuItems** table.

```
ALTER TABLE MenuItems MODIFY popularity INT(11) COMMENT 'Popularity based on 2019 sales';
```

Using ORDER BY Clause

Consider the following **Places** table:

List all the regions and cities in Canada, but list them by region in reverse alphabetical order.


```
SELECT Region, City FROM Places WHERE Country = 'Canada' ORDER BY Region DESC;
```

Displaying Data in Alphabetical Order

Consider the following **Places** table:

List the region and city in the United States, but make the region list descending and the cities within each region ascending.

```
SELECT Region, City FROM Places WHERE Country = 'United States' ORDER BY Region DESC, City ASC;
```

Using GROUP BY Clause

Consider the following **Places** table:

Write a query to retrieve countries and count how many cities are in each country using a GROUP BY clause.

```
SELECT Country, COUNT(City) FROM Places GROUP BY Country;
```

Using COUNT Function in GROUP BY Clause

Consider the following **Places** table:

Write a query to retrieve region and do a count of all the cities (city) in each region, but only for the United States.

```
SELECT Region, COUNT(City) FROM Places WHERE Country = 'United States' GROUP BY Region;
```

Using Operators in GROUP BY Clause

List all the regions that have more than two cities in them.

```
SELECT Region FROM Places GROUP BY Region HAVING COUNT(City)>2;
```

Using DISTINCT Keyword

Consider the following **Things** table:

Select (distinctly) all of the colors from the **Things** table.

```
SELECT DISTINCT color FROM Things;
```

Using ORDER BY Clause

List all the names and weights of animals in species 2, but list them by weights in reverse numerical order.

```
SELECT Name, Weight FROM Animals  
WHERE Species = 2  
ORDER BY Weight DESC;
```

Displaying Data in Alphabetical Order

List the make and model of all Chevrolets and Jaguars, but arrange the make list to be descending and the model of each make to be ascending.

```
SELECT make, model FROM garage  
WHERE make = 'Chevrolet' OR make = "Jaguar"  
ORDER BY make DESC, model ASC;
```

Using GROUP BY Clause

Question 67 :

Consider the following **garage** table:

Write a query to retrieve makes and count how many models are in each make using a GROUP BY clause.

```
SELECT make, COUNT(model) FROM garage GROUP BY make;
```

Using COUNT Function in GROUP BY Clause

Write a query to retrieve the make and do a count of all the models of each make with model year after 1950.

```
SELECT make, COUNT(model) FROM garage  
WHERE year > 1950  
GROUP BY make;
```

Using **Operators** in GROUP BY Clause

List all the makes that have more than 3 models in them.

```
SELECT make FROM garage
GROUP BY make
HAVING COUNT(model)>3;
```

Using **DISTINCT** Keyword

Select (distinctly) all of the makes from the **garage** table.

```
SELECT DISTINCT make FROM garage;
```

Joining Tables in MySQL

Write a query in the **MySQL Editor** that joins (**Inner Join**) the three tables given below and meets the following requirements:

- Retrieve **PizzaOrderID, LastName, PizzaName, OrderQty, PizzaPrice** from the following tables.
- Use the table name alias as **o** for **uc_pizzaorder**, **p** for **uc_pizza**, and **c** for **uc_customer** tables.

```
• SELECT o.PizzaOrderID,
  c.LastName,
  p.PizzaName,
  o.OrderQty,
  p.PizzaPrice
FROM uc_pizzaorder As o
INNER JOIN uc_pizza AS p ON p.PizzaID = o.PizzaID
INNER JOIN uc_customer AS c ON c.CustomerID = o.CustomerID;
```

Joining Tables in MySQL

Write a query in the MySQL Editor that joins (Inner Join) the three tables given below and meets the following requirements:

- Retrieve DonutOrderID, LastName, DonutName, OrderQty, DonutPrice from the following tables.

- Use the table name alias as **o** for **uc_donutorder**, **d** for **uc_donut**, and **c** for **uc_customer** tables.

Structure of the required tables is given below:

```

• SELECT o.DonutOrderID,
• c.LastName,
• d.DonutName,
• o.OrderQty,
• d.DonutPrice
• FROM uc_donutorder As o
• INNER JOIN uc_donut AS d ON d.DonutID = o.DonutID
• INNER JOIN uc_customer AS c ON c.CustomerID = o.CustomerID;

```

Using CONCAT Function

Select each employees first_name and last_name and concatenate them together to form the first column. Then, select salary column and divide it by 12 to get the monthly salary for each employee. Finally, call that second column as monthly_cost so we can use it later.

```

SELECT CONCAT(first_name, " ", last_name), (salary/12) AS monthly_cost FROM Employees;

```

Using CONCAT Function

Select the birth_year, birth_month and birth_day and concatenate them with a hyphen(-) to form a column named birthday. Then select the salary column and multiply it by 10% to get the maximum commission for each employee. Call this column max_commission.

```

SELECT CONCAT(birth_year, "-", birth_month, "-", birth_day) AS birthday, (salary*.10) AS max_commission FROM Employees;

```

Create a View in MySQL

Create a view in MySQL Editor that contains the following information:

For the given customer table, use the appropriate **CREATE VIEW V_Customer AS SELECT FROM Customer;** command so that it concatenates the FirstName, with a blank space and the LastName fields to create an attribute CustomerName. Show all other columns in the view as they are.

Hint: Use this command to combine first name and last name:

CONCAT(FirstName , ' ',LastName) AS CustomerName

```
CREATE VIEW V_Customer
AS SELECT CustomerID,
CONCAT(FirstName , ' ',LastName) AS CustomerName,
Street,
Apt,
City,
State,
Zip,
HomePhone,
MobilePhone,
OtherPhone
FROM Customer;
```

Creating a View

Create a view called **BirthdaySquad** that only include first_name, last_name, birth_month, and birth_day from the **Employees** table.

```
CREATE VIEW BirthdaySquad AS SELECT first_name, last_name, birth
_month, birth_day FROM Employees;
```

Creating a View Using UNION Operator

Create a **BurnRate** view that will achieve the following tasks and combine them using the UNION operator.

- Select each employees first_name and last_name and concatenate them together to form the first column. Then, select the salary column

as the second column and divide it by 12 to get the monthly salary for each employee and call this second column as monthly_cost.

- Select building_name, monthly_rent from the **Buildings** table.
- SELECT company, monthly_fee from the **Services** table.
- ```
CREATE VIEW BurnRate AS
SELECT CONCAT(first_name, " ", last_name), (salary/12) AS monthly_cost FROM Employees
UNION
SELECT building_name, monthly_rent FROM Buildings
UNION
SELECT company, monthly_fee FROM Services;
```

## Create a View in MySQL

Create a view in MySQL Editor that contains the following information:

For the given **Employees** table, use the appropriate **CREATE VIEW V\_Commission AS SELECT FROM Employees;** command so that it concatenates the first\_name and last\_name with a blank space to create an attribute customer\_name. Then it computes 10% of the salary to create an attribute commission. Finally it concatenates the birth\_year, birth\_month and birthday with a hyphen(-) to create an attribute birthday. Show all other columns in the view as they are.

The resulting view will display data with these column headings:

| id | customer_name | address | comission | birth |
|----|---------------|---------|-----------|-------|
|----|---------------|---------|-----------|-------|

- ```
CREATE VIEW V_Commission
```
- ```
AS SELECT id,
```
- ```
CONCAT(first_name, ' ', last_name) AS customer_name,
```
- ```
address,
```
- ```
(salary * .10) AS commission,
```
- ```
CONCAT(birth_year, '-', birth_month, '-', birth_day) AS birthday FROM Employees;
```

## Creating a View

Create a view called **LargeAnimals** that only includes Name and Weight for all animals over 100 from the **Animals** table.

- `CREATE VIEW LargeAnimals AS SELECT Name, Weight FROM Animals WHERE Weight > 100;`

## Creating a View Using the UNION Operator

Create a view called **MyDates** that will achieve the following tasks and combine them using the UNION operator.

- From the **Employees** table, select last\_name and first\_name and concatenate them together using a comma(,) and blank space to form the column named names. Then select birth\_year, birth\_month and birth\_day and concatenate them together using a hyphen(-) to form the column named dates.
- Select Name and Purchased from the **Albums** table.
- Select Name and LastWeighDate from the **Animals** table for all animals weighing more than 100.
- Order the **MyDates** view by the names column.

```
CREATE VIEW MyDates AS
SELECT CONCAT(last_name, ", ", first_name) AS names, CONCAT
(birth_year, "-", birth_month, "-", birth_day) AS dates FROM
Employees
UNION
SELECT Name, Purchased FROM Albums
UNION
SELECT Name, LastWeighDate FROM Animals WHERE Weight > 100
ORDER BY names;
```

## Create an Index in MySQL

Create an index named as **i\_PizzaName**, which is indexing the **PizzaName** column of the **Pizza** table by using the **CREATE INDEX** index\_name **ON Pizza** ( column1, column2,...); command in MySQL Editor.

```
CREATE INDEX I_PizzaName ON Pizza (PizzaName);
```

## Creating an Index in MySQL

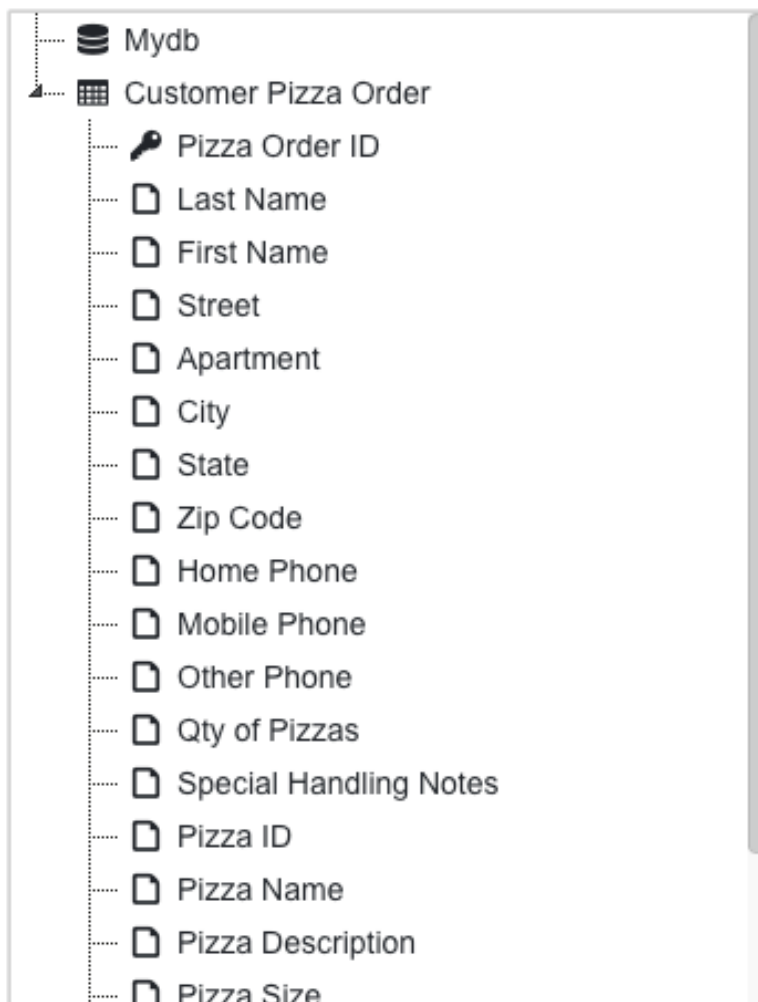
Create an index named **I\_EEName**, which is indexing the last\_name and then first\_name columns of the **Employees** table by using the **CREATE INDEX**

index\_name **ON Employees** (column1, column2,...); command in MySQL Editor.

```
CREATE INDEX I_EEName ON Employees (last_name, first_name);
```

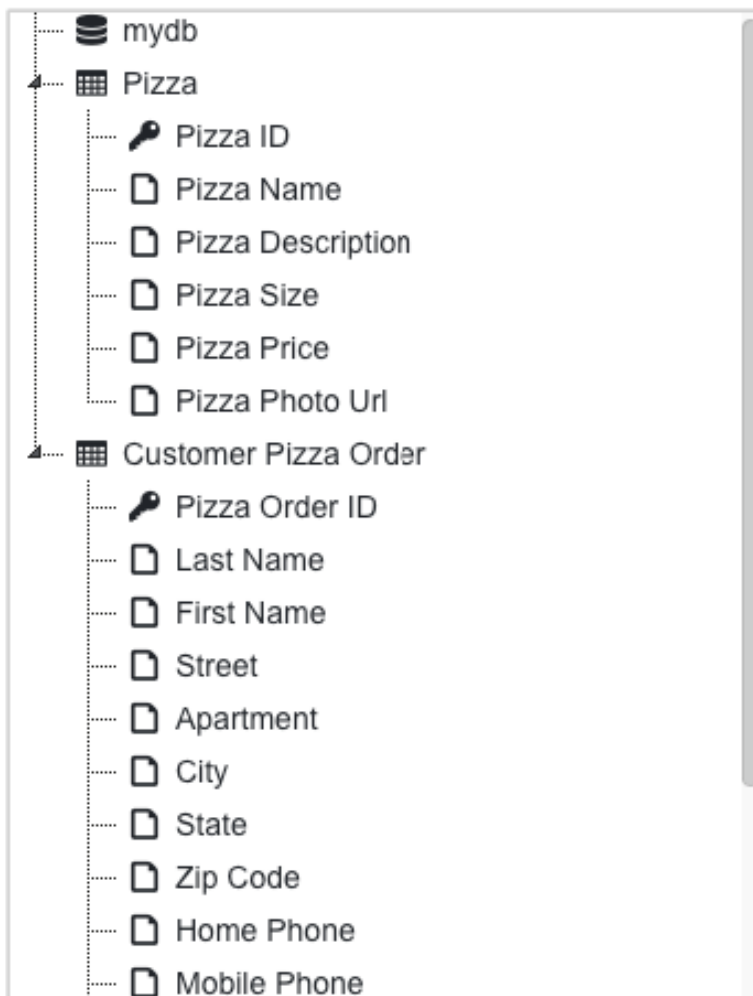


# Entities

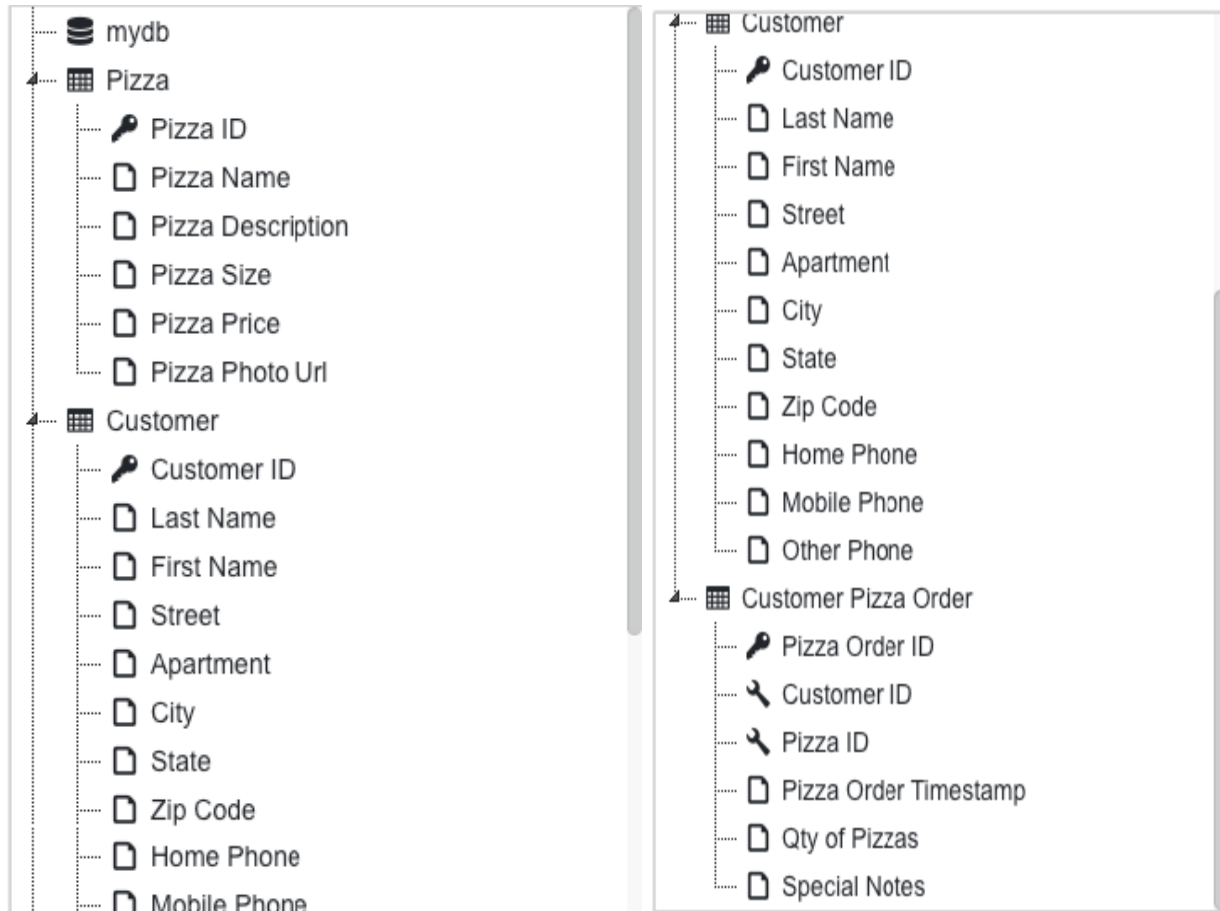


1NF

# Entities



2NF



### 3NF

#### Creating a Procedure

Create a procedure named `count_rating` to return the count of Albums based on the user input of the Rating level of Artists.

| ID | Name                | Description                | Rating |
|----|---------------------|----------------------------|--------|
| 1  | Cross Join Blues    | Weird SQL band             | 1      |
| 2  | No Indexes          | Slow and steady            | 3      |
| 3  | Join In             | SQL themed singalong group | 2      |
| 4  | No Where Query      | Sings about everything     | 1      |
| 5  | See Quells          | Another bad pun band       | 2      |
| 6  | Computers for Games | Gamer oriented band        | 1      |

Table A: Artists

| ID | Artist(ID) | Name                  | Notes                      | Media | Cost  | Sealed | Purchased  |
|----|------------|-----------------------|----------------------------|-------|-------|--------|------------|
| 1  | 1          | Wild Data             |                            | 4     | 19.95 | Y      | 1/9/2010   |
| 2  | 2          | Up In The Cloud       |                            | 1     | 5     | Y      | 6/26/1999  |
| 3  | 2          | Big Data For All      | Autographed!!!             | 1     | 0     | N      | 7/3/2018   |
| 4  | 3          | Processing for Cash   |                            | 2     | 6.95  | N      | 1/5/2018   |
| 5  | 4          | Network News          | Pretty bad song collection | 3     | 9.99  | N      | 6/23/2017  |
| 6  | 5          | Living for the Cloud  |                            | 4     | 10    | N      | 11/20/2017 |
| 7  | 1          | 1 2 3!                | Loaned to Sam              | 1     | 5     | Y      | 2/7/2016   |
| 8  | 2          | Waiting on Disk       |                            | 2     | 22.99 | N      | 12/6/2017  |
| 9  | 3          | QUIET!: The Anthology |                            | 1     | 3     | N      | 8/8/2017   |
| 10 | 1          | A Bug or Feature?     |                            | 4     | 2     | Y      | 10/31/2017 |
| 11 | 2          | Waiting on Disk       |                            | 2     | 22.99 | N      | 12/6/2017  |

Table B: Albums

```

DELIMITER //
CREATE PROCEDURE count_rating (IN var1 INT)
BEGIN
select count(*) from Artists a, Albums b where a.Rating = var1 a
nd a.ID = b.Artist;
END//

```