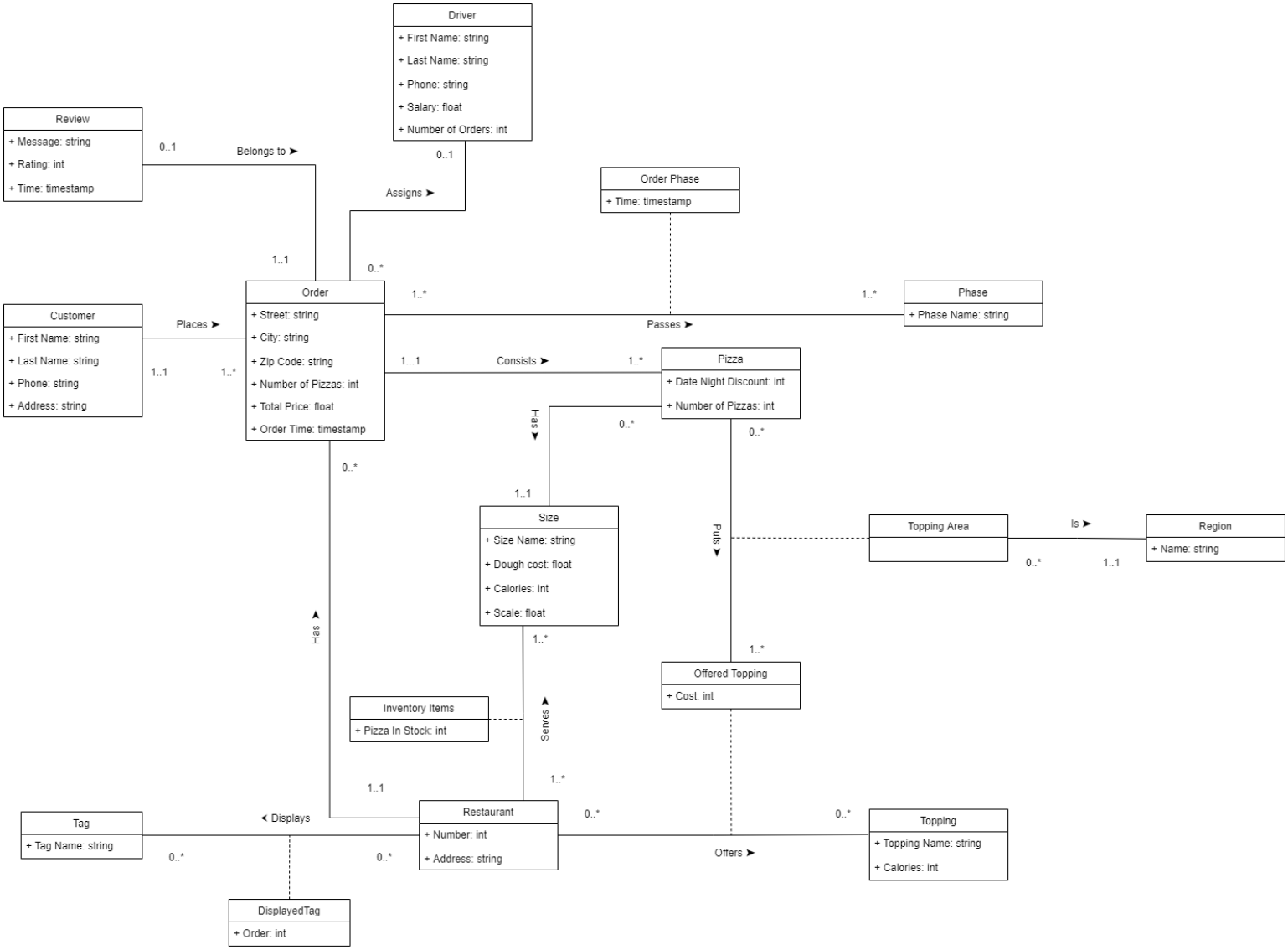
CECS 323
Introduction to Databases

Project 02 – Cowabunga!

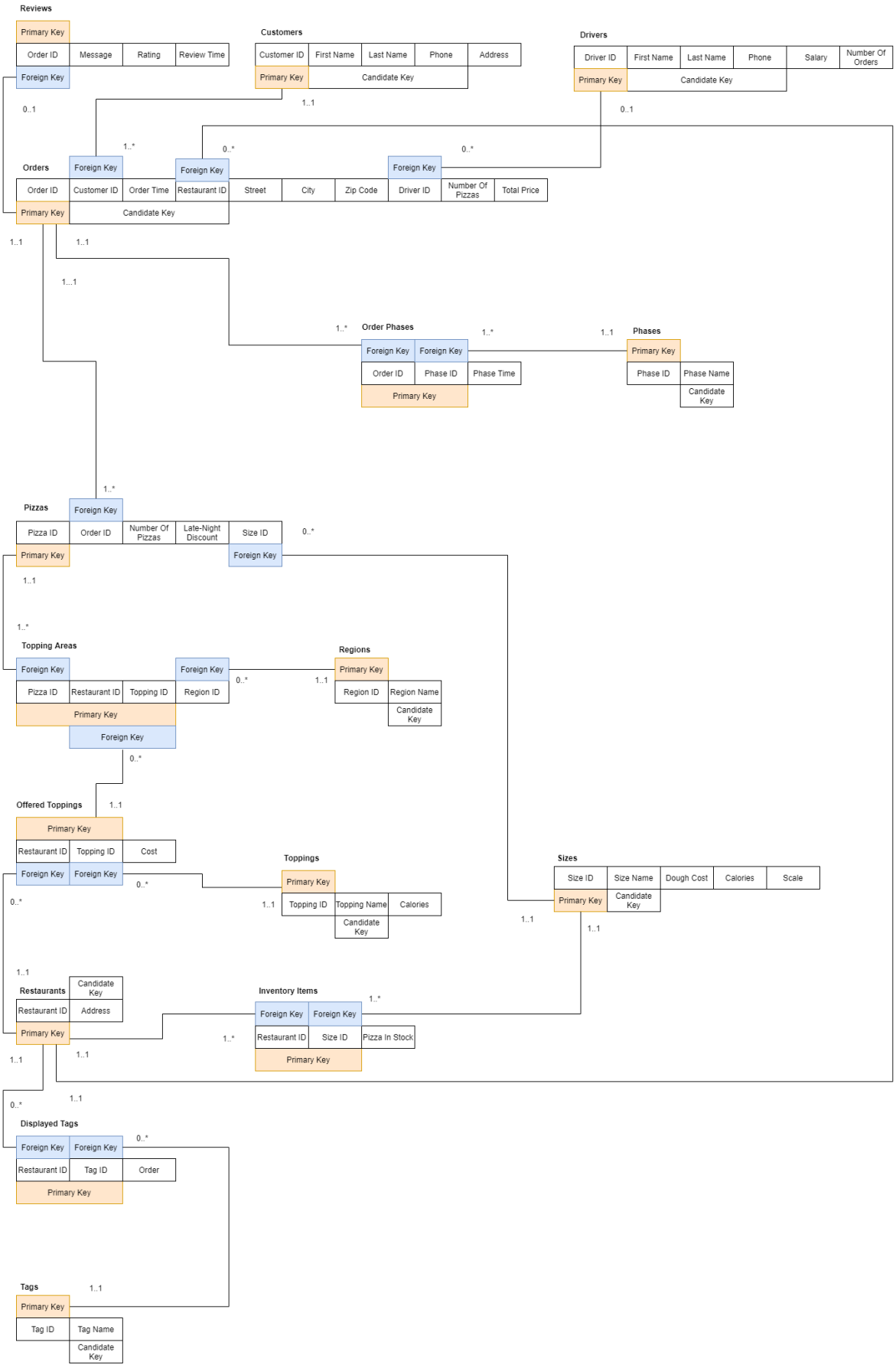
Team Members: Christopher Shih – Thanh Nguyen

Due date: October 27 at 2:00pm

I. UML diagram:



II. Relation scheme diagram:


Reviews
Primary Key
Order ID
Message
Rating
Review Time
Foreign Key
Customers
Customer ID
First Name
Last Name
Phone
Address
Primary Key
Candidate Key
Drivers
Driver ID
First Name
Last Name
Phone
Salary
Number Of Orders
Primary Key
Candidate Key
0..1
1..1
0..1
1..*
0..*
0..*
Orders
Foreign Key
Foreign Key
Foreign Key
Order ID
Customer ID
Order Time
Restaurant ID
Street
City
Zip Code
Driver ID
Number Of Pizzas
Total Price
Primary Key
Candidate Key
1..1
1..1
1...1
1..*
Order Phases
1..*
1..1
Phases
Foreign Key
Foreign Key
Primary Key
Order ID
Phase ID
Phase Time
Primary Key
Phase ID
Phase Name
Candidate Key
1..*
Pizzas
Foreign Key
Pizza ID
Order ID
Number Of Pizzas
Late-Night Discount
Size ID
Primary Key
Foreign Key
0..*
1..1
1..*
Topping Areas
Regions
Foreign Key
Foreign Key
Primary Key
Pizza ID
Restaurant ID
Topping ID
Region ID
Region ID
Region Name
Primary Key
Candidate Key
Foreign Key
0..*
0..*
1..1
0..*
Offered Toppings
1..1
Primary Key
Restaurant ID
Topping ID
Cost
Foreign Key
Foreign Key
0..*
0..*
Toppings
Primary Key
Topping ID
Topping Name
Calories
Candidate Key
1..1
Sizes
Size ID
Size Name
Dough Cost
Calories
Scale
Primary Key
Candidate Key
1..1
1..1
1..1
Restaurants
Candidate Key
Restaurant ID
Address
Primary Key
Inventory Items
Foreign Key
Foreign Key
Restaurant ID
Size ID
Pizza In Stock
Primary Key
1..*
1..*
1..1
1..1
0..*
1..1
Displayed Tags
Foreign Key
Foreign Key
Restaurant ID
Tag ID
Order
Primary Key
0..*
Tags
1..1
Primary Key
Tag ID
Tag Name
Candidate Key

III. DDL commands to generate tables in DataGrip/Derby:

1. CUSTOMERS table

```
create table CUSTOMERS
(
    "CustomerID" INTEGER default AUTOINCREMENT: start 1 increment 1 generated
always as identity
        constraint CUSTOMERS_PK
        primary key,
    "FirstName"  VARCHAR(20) not null,
    "LastName"   VARCHAR(20) not null,
    "Phone"      CHAR(10)    not null,
    "Address"    VARCHAR(50),
    constraint CUSTOMERS_FIRSTNAME_LASTNAME_PHONE_UINDEX
        unique ("FirstName", "LastName", "Phone")
);
```

2. DRIVERS table

```
create table DRIVERS
(
    "DriverID"       INTEGER default AUTOINCREMENT: start 1 increment 1 generated
always as identity
        constraint DRIVERS_PK
        primary key,
    "FirstName"      VARCHAR(20) not null,
    "LastName"       VARCHAR(20) not null,
    "Phone"          CHAR(10)    not null,
    "Salary"         DOUBLE      not null,
    "NumberOfOrders" INTEGER     not null,
    constraint DRIVERS_FIRSTNAME_LASTNAME_PHONE_UINDEX
        unique ("FirstName", "LastName", "Phone")
);
```

3. PHASES table

```
create table PHASES
(
    "PhaseID"   INTEGER default AUTOINCREMENT: start 1 increment 1 generated always
as identity
        constraint PHASES_PK
        primary key,
    "PhaseName" VARCHAR(20) not null
        constraint PHASES_PHASENAME_UINDEX
            unique
);
```

4. REGIONS table

```
create table REGIONS
(
    "RegionID"   INTEGER default AUTOINCREMENT: start 1 increment 1 generated
always as identity
        constraint REGIONS_PK
```

```
        primary key,
    "RegionName" VARCHAR(20) not null
        constraint REGIONS_REGIONNAME_UINDEX
            unique
);
```

5. TOPPINGS table

```
create table TOPPINGS
(
    "ToppingID"   INTEGER default AUTOINCREMENT: start 1 increment 1 generated
always as identity
        constraint TOPPINGS_PK
        primary key,
    "ToppingName" VARCHAR(20) not null
        constraint TOPPINGS_TOPPINGNAME_UINDEX
            unique,
    "Calories"    INTEGER     not null
);
```

6. SIZES table

```
create table SIZES
(
    "SizeID"    INTEGER default AUTOINCREMENT: start 1 increment 1 generated always
as identity
        constraint SIZES_PK
        primary key,
    "SizeName"  VARCHAR(20) not null
        constraint SIZES_SIZENAME_UINDEX
            unique,
    "DoughCost" DOUBLE      not null,
    "Calories"  INTEGER     not null,
    "Scale"      DOUBLE       not null
);
```

7. RESTAURANTS table

```
create table RESTAURANTS
(
    "RestaurantID" INTEGER default AUTOINCREMENT: start 1 increment 1 generated
always as identity
        constraint RESTAURANTS_PK
        primary key,
    "Address"      VARCHAR(50) not null
        constraint RESTAURANTS_ADDRESS_UINDEX
            unique
);
```

8. TAGS table

```
create table TAGS
(
    "TagID"   INTEGER default AUTOINCREMENT: start 1 increment 1 generated always
as identity
        constraint TAGS_PK
        primary key,
    "TagName" VARCHAR(20) not null
        constraint TAGS_TAGNAME_UINDEX
```

```
            unique
);
```

9. DISPLAYEDTAGS table

```
create table DISPLAYEDTAGS
(
    "RestaurantID" INTEGER not null
        constraint DISPLAYEDTAGS_RESTAURANTS_RESTAURANTID_FK
            references RESTAURANTS,
    "TagID"        INTEGER not null
        constraint DISPLAYEDTAGS_TAGS_TAGID_FK
            references TAGS,
    "Order"        INTEGER not null,
    constraint DISPLAYEDTAGS_PK
        primary key ("RestaurantID", "TagID")
);
```

10. INVENTORYITEMS table

```
create table INVENTORYITEMS
(
    "RestaurantID" INTEGER not null
        constraint INVENTORYITEMS_RESTAURANTS_RESTAURANTID_FK
            references RESTAURANTS,
    "SizeID"       INTEGER not null
        constraint INVENTORYITEMS_SIZES_SIZEID_FK
            references SIZES,
    "PizzaInStock" INTEGER not null,
    constraint INVENTORYITEMS_PK
        primary key ("RestaurantID", "SizeID")
);
```

11. OFFEREDTOPPINGS table

```
create table OFFEREDTOPPINGS
(
    "RestaurantID" INTEGER not null
        constraint OFFEREDTOPPINGS_RESTAURANTS_RESTAURANTID_FK
            references RESTAURANTS,
    "ToppingID"    INTEGER not null
        constraint OFFEREDTOPPINGS_TOPPINGS_TOPPINGID_FK
            references TOPPINGS,
    "Cost"         DOUBLE  not null,
    constraint OFFEREDTOPPINGS_PK
        primary key ("RestaurantID", "ToppingID")
);
```

12. ORDERS table

```
create table ORDERS
(
    "OrderID"        INTEGER default AUTOINCREMENT: start 1 increment 1 generated
always as identity
        constraint ORDERS_PK
        primary key,
    "CustomerID"     INTEGER   not null
        constraint ORDERS_CUSTOMERS_CUSTOMERID_FK
            references CUSTOMERS,
```

```
    "OrderTime"      TIMESTAMP not null,
    "Street"         VARCHAR(30),
    "City"           VARCHAR(20),
    "ZipCode"        CHAR(5),
    "NumberOfPizzas" INTEGER   not null,
    "TotalPrice"     DOUBLE,
    "DriverID"       INTEGER
        constraint ORDERS_DRIVERS_DRIVERID_FK
            references DRIVERS,
    "RestaurantID"   INTEGER   not null
        constraint ORDERS_RESTAURANTS_RESTAURANTID_FK
            references RESTAURANTS,
    constraint ORDERS_CUSTOMERID_ORDERTIME_RESTAURANTID_UINDEX
        unique ("CustomerID", "OrderTime", "RestaurantID")
);
```

13. REVIEWS table

```
create table REVIEWS
(
    "OrderID"    INTEGER      not null
        constraint REVIEWS_PK
            primary key
        constraint REVIEWS_ORDERS_ORDERID_FK
            references ORDERS,
    "Message"    VARCHAR(100) not null,
    "Rating"     INTEGER      not null,
    "ReviewTime" TIMESTAMP    not null
);
```

14. ORDERPHASES table

```
create table ORDERPHASES
(
    "OrderID"   INTEGER   not null
        constraint ORDERPHASES_ORDERS_ORDERID_FK
            references ORDERS,
    "PhaseID"   INTEGER   not null
        constraint ORDERPHASES_PHASES_PHASEID_FK
            references PHASES,
    "PhaseTime" TIMESTAMP not null,
    constraint ORDERPHASES_PK
        primary key ("OrderID", "PhaseID")
);
```

15. PIZZAS table

```
create table PIZZAS
(
    "PizzaID"            INTEGER default AUTOINCREMENT: start 1 increment 1
generated always as identity
        constraint PIZZAS_PK
        primary key,
    "OrderID"            INTEGER not null
        constraint PIZZAS_ORDERS_ORDERID_FK
            references ORDERS,
    "NumberOfPizzas"     INTEGER not null,
    "Late-NightDiscount" DOUBLE,
    "SizeID"             INTEGER not null
```

```
        constraint PIZZAS_SIZES_SIZEID_FK
            references SIZES
);
```

16. TOPPINGAREAS table

```
create table TOPPINGAREAS
(
    "PizzaID"      INTEGER not null
        constraint TOPPINGAREAS_PIZZAS_PIZZAID_FK
            references PIZZAS,
    "RestaurantID" INTEGER not null,
    "ToppingID"    INTEGER not null,
    "RegionID"     INTEGER not null
        constraint TOPPINGAREAS_REGIONS_REGIONID_FK
            references REGIONS,
    constraint TOPPINGAREAS_PK
        primary key ("PizzaID", "RestaurantID", "ToppingID"),
    constraint TOPPINGAREAS_OFFEREDTOPPINGS_RESTAURANTID_TOPPINGID_FK
        foreign key ("RestaurantID", "ToppingID") references PIZZAS (RestaurantID,
ToppingID)
);
```

IV. Numbered answers to the required SQL SELECT statements:

1. **Given a specific pizza, display all toppings chosen for that pizza, including their region. Sort them so the whole-pizza toppings come first, then the left half, then the right half; and order within each region alphabetically.**

```
SELECT P."PizzaID", "ToppingName", "RegionName"
FROM PIZZAS P
    INNER JOIN TOPPINGAREAS A ON P."PizzaID" = A."PizzaID"
    INNER JOIN TOPPINGS T ON A."ToppingID" = T."ToppingID"
    INNER JOIN REGIONS R ON R."RegionID" = A."RegionID"
WHERE P."PizzaID" = 1
ORDER BY
        CASE
            WHEN LOWER(R."RegionName") = 'whole pizza' THEN 1
            WHEN LOWER(R."RegionName") = 'left half' THEN 2
            WHEN LOWER(R."RegionName") = 'right half' THEN 3
        END,
        "ToppingName";
```

Example output:

	"PizzaID"	"ToppingName"	"RegionName"
1	1	Cheese	Whole Pizza
2	1	Mushroom	Whole Pizza
3	1	Pineapple	Left Half
4	1	Ham	Right Half
5	1	Pepperoni	Right Half

2. **Given a specific order, display the total cost and total calories of each pizza in that order.**

```
SELECT P."PizzaID", SUM(T."Calories" * S."Scale") + S."Calories" AS Calories,
SUM(OT."Cost" * S."Scale") + S."DoughCost" AS Cost
FROM ORDERS O
    INNER JOIN PIZZAS P ON O."OrderID" = P."OrderID"
    INNER JOIN TOPPINGAREAS A ON P."PizzaID" = A."PizzaID"
    INNER JOIN OFFEREDTOPPINGS OT ON OT."RestaurantID" = A."RestaurantID" AND
OT."ToppingID" = A."ToppingID"
    INNER JOIN TOPPINGS T ON T."ToppingID" = OT."ToppingID"
    INNER JOIN SIZES S ON P."SizeID" = S."SizeID"
WHERE O."OrderID" = 1
GROUP BY P."PizzaID", S."Calories", S."DoughCost";
```

Example output:

	PizzaID	CALORIES	COST
1	1	90	27
2	2	45	12
3	3	49	14.2

3. **Show all restaurant identifiers (chain numbers) and their average Review rating for orders handled by that restaurant.**

```
SELECT R."RestaurantID", R."Address", AVG(CAST(RV."Rating" AS float)) AS
AverageRating
FROM RESTAURANTS R
    LEFT JOIN ORDERS O ON R."RestaurantID" = O."RestaurantID"
    LEFT JOIN REVIEWS RV ON O."OrderID" = RV."OrderID"
GROUP BY R."RestaurantID", R."Address";
```

Example output:

	RestaurantID	Address	AVERAGERATING
1	1	4897 Euclid St, Anaheim CA 92801	4.25
2	2	10547 Katella Ave, Garden Grove CA 92804	<null>
3	3	8745 La Palma Ave, Buena Park CA 90620	2

4. **Count the number of pizzas that do not have pepperoni as a topping.**

```
SELECT COUNT(DISTINCT "PizzaID") AS NumberOfPizzas
FROM PIZZAS
WHERE "PizzaID" NOT IN
     (SELECT A."PizzaID"
      FROM TOPPINGAREAS A INNER JOIN TOPPINGS T ON A."ToppingID" = T."ToppingID"
      WHERE "ToppingName" = 'Pepperoni');
```

5. **Show all customer names who submitted a Review with a rating of 4 or greater for an Order with at least one pizza that has pepperoni as one of its ingredients. Sort the names alphabetically.**

```
SELECT DISTINCT "LastName", "FirstName"
FROM CUSTOMERS C
    INNER JOIN ORDERS O ON C."CustomerID" = O."CustomerID"
    INNER JOIN PIZZAS P ON O."OrderID" = P."OrderID"
    INNER JOIN REVIEWS R ON O."OrderID" = R."OrderID"
WHERE R."Rating" >= 4 AND P."PizzaID" IN
     (SELECT "PizzaID"
      FROM TOPPINGAREAS A INNER JOIN TOPPINGS T ON A."ToppingID" = T."ToppingID"
      WHERE "ToppingName" = 'Pepperoni')
ORDER BY "LastName", "FirstName";
```

6. **Identify the pizza that has the highest calorie total -- the sum of its dough and toppings -- of any pizza in the database.**

```
SELECT P."PizzaID", S."Calories" + SUM(T."Calories" * S."Scale") AS TotalCalories
FROM PIZZAS P
    INNER JOIN TOPPINGAREAS A ON P."PizzaID" = A."PizzaID"
    INNER JOIN TOPPINGS T ON T."ToppingID" = A."ToppingID"
    INNER JOIN SIZES S on P."SizeID" = S."SizeID"
GROUP BY P."PizzaID", S."Calories"
HAVING S."Calories" + SUM(CAST(T."Calories" AS float) * S."Scale") >=
       ALL(SELECT S2."Calories" + SUM(CAST(T2."Calories" AS float) * S2."Scale")
           FROM PIZZAS P2
               INNER JOIN TOPPINGAREAS A2 ON P2."PizzaID" = A2."PizzaID"
               INNER JOIN TOPPINGS T2 ON T2."ToppingID" = A2."ToppingID"
```

```
        INNER JOIN SIZES S2 on P2."SizeID" = S2."SizeID"
GROUP BY P2."PizzaID", S2."Calories");
```

Example Output:

	PizzaID	TOTALCALORIES
1	4	100
2	11	100