**A1.**

**Second Normal Form (2NF)**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **BAGEL ORDER** | |  | **BAGEL ORDER LINE ITEM** | |  | **BAGEL** | |
| PK | Bagel Order ID |  | PK / FK | Bagel Order ID |  | PK | Bagel ID |
|  | Order Date | 1:M | PK / FK | Bagel ID | M:1 |  | Bagel Name |
|  | First Name |  |  | Bagel Quantity |  |  | Bagel Description |
|  | Last Name |  |  |  |  |  | Bagel Price |
|  | Address 1 |  |  |  |  |  |  |
|  | Address 2 |  |  |  |  |  |  |
|  | City |  |  |  |  |  |  |
|  | State |  |  |  |  |  |  |
|  | Zip |  |  |  |  |  |  |
|  | Mobile Phone |  |  |  |  |  |  |
|  | Delivery Fee |  |  |  |  |  |  |
|  | Special Notes |  |  |  |  |  |  |

I assigned each attribute by finding the functional dependence of each one and determining whether all non-key attributes were dependent on the whole Primary Key. In the 1NF table, the attributes Bagel Name, Bagel Description, and Bagel Price depend on Bagel ID alone. Since the Primary Key is a composite of the Bagel ID and the Bagel Order ID, this does not satisfy the requirements of 2NF. To resolve this, we merely create a new table of these attributes with Bagel ID as the Primary Key. Unfortunately, this does not completely resolve the problem because all the other attributes (except for Bagel Quantity) depend on Bagel Order ID. We can resolve this the same way as before by creating a new table with Bagel Order ID as the Primary Key. This leaves us with three tables where all tables are in 2NF and each attribute that is not a Primary Key depends on the whole Primary Key.

To determine the cardinality, I analyzed the relationship between tables. For example, A specific bagel order can have multiple bagel order line items. The relationship between BAGEL ORDER and BAGEL ORDER LINE ITEM is one-to-many because each value of Bagel Order ID occurs once in the first table and many times in the second table. Therefore, for each BAGEL ORDER, there are many BAGEL ORDER LINE ITEMS. Also, in the BAGEL ORDER LINE TABLE the same Bagel ID value can be repeated multiple times while in the BAGEL table, it cannot. This relationship is therefore many-to-one.

**A2.**

**Third Normal Form (3NF)**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **BAGEL ORDER** | | |  | **BAGEL ORDER LINE ITEM** | |  | **BAGEL** | |
| PK | Bagel Order ID | |  | PK / FK | Bagel Order ID |  | PK | Bagel ID |
| FK | Customer ID | | 1:M | PK / FK | Bagel ID | M:1 |  | Bagel Name |
|  | Order Date | |  |  | Bagel Quantity |  |  | Bagel Description |
|  | Delivery Fee | |  |  |  |  |  | Bagel Price |
|  | Special Notes | |  |  |  |  |  |  |
|  | M:1 |  |  |  |  |  |  |  |
| **CUSTOMER** | | |  |  |  |  |  |  |
| PK | Customer ID | |  |  |  |  |  |  |
|  | First Name | |  |  |  |  |  |  |
|  | Last Name | |  |  |  |  |  |  |
|  | Address 1 | |  |  |  |  |  |  |
|  | Address 2 | |  |  |  |  |  |  |
|  | City | |  |  |  |  |  |  |
|  | State | |  |  |  |  |  |  |
|  | Zip | |  |  |  |  |  |  |
|  | Mobile Phone | |  |  |  |  |  |  |

I assigned each attribute by finding the functional dependence of each one and determining whether no non-key attributes were transitively dependent on other non-key attributes. Because the Primary Key in the BAGEL ORDER table wasn’t the only candidate key that the customer information could depend on, we removed it, creating a new table, CUSTOMER, with a new attribute, Customer ID, as its Primary Key. We then used this Key as a Foreign Key within the BAGEL ORDER table, forming a relationship between this table and the table we just created. Now all attributes depend on the Key, the whole Key, and nothing but the Key of each table.

I determined the cardinality of this new table’s relationship by analyzing its relationship to its corresponding table, BAGEL ORDER. Because every bagel order has a maximum of one customer, but a single customer can have many orders, the relationship between BAGEL ORDER and CUSTOMER, respectively, is many-to-one. The relationships between the other tables remain the same as they were in the 2NF form.

**A3.**

**Final Physical Database Model**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **BAGEL ORDER** | | |  | **BAGEL ORDER LINE ITEM** | | |  | **BAGEL** | |  |
| PK | bagel\_order\_id | INT |  | PK / FK | bagel\_order\_id | INT |  | PK | bagel\_id | CHAR(2) |
| FK | customer\_id | INT | 1:M | PK / FK | bagel\_id | CHAR(2) | M:1 |  | bagel\_name | VARCHAR(30) |
|  | order\_date | TIMESTAMP |  |  | bagel\_quantity | INT |  |  | bagel\_description | VARCHAR(50) |
|  | delivery\_fee | NUMERIC(5, 2) |  |  |  |  |  |  | bagel\_price | NUMERIC(5, 2) |
|  | special\_notes | VARCHAR(100) |  |  |  |  |  |  |  |  |
|  | M:1 |  |  |  |  |  |  |  |  |  |
| **CUSTOMER** | | |  |  |  |  |  |  |  |  |
| PK | customer\_id | INT |  |  |  |  |  |  |  |  |
|  | first\_name | VARCHAR(30) |  |  |  |  |  |  |  |  |
|  | last\_name | VARCHAR(30) |  |  |  |  |  |  |  |  |
|  | address1 | VARCHAR(30) |  |  |  |  |  |  |  |  |
|  | address2 | VARCHAR(30) |  |  |  |  |  |  |  |  |
|  | city | VARCHAR(30) |  |  |  |  |  |  |  |  |
|  | state | CHAR(2) |  |  |  |  |  |  |  |  |
|  | zip | VARCHAR(7) |  |  |  |  |  |  |  |  |
|  | mobile\_phone | VARCHAR(14) |  |  |  |  |  |  |  |  |

**B1.**

CREATE TABLE COFFEE\_SHOP (

shop\_id INTEGER,

shop\_name VARCHAR(50),

city VARCHAR(50),

state CHAR(2),

PRIMARY KEY (shop\_id)

);

CREATE TABLE EMPLOYEE (

employee\_id INTEGER,

first\_name VARCHAR(30),

last\_name VARCHAR(30),

hire\_date DATE,

job\_title VARCHAR(30),

shop\_id INTEGER,

PRIMARY KEY (employee\_id),

FOREIGN KEY (shop\_id) REFERENCES COFFEE\_SHOP(shop\_id)

);

CREATE TABLE SUPPLIER (

supplier\_id INTEGER,

company\_name VARCHAR(50),

country VARCHAR(30),

sales\_contact\_name VARCHAR(60),

email VARCHAR(50) NOT NULL,

PRIMARY KEY (supplier\_id)

);

CREATE TABLE COFFEE (

coffee\_id INTEGER,

shop\_id INTEGER,

supplier\_id INTEGER,

coffee\_name VARCHAR(30),

price\_per\_pound NUMERIC(5,2),

PRIMARY KEY (coffee\_id),

FOREIGN KEY (shop\_id) REFERENCES COFFEE\_SHOP(shop\_id),

FOREIGN KEY (supplier\_id) REFERENCES SUPPLIER(supplier\_id)

);

A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated

**B2.**

INSERT INTO COFFEE\_SHOP (shop\_id, shop\_name, city, state) VALUES

(1, 'Starbucks', 'Chicago', 'IL'),

(2, 'Coffee Oasis', 'Seattle', 'WA'),

(3, 'The Grind', 'Richmond', 'VA');

INSERT INTO EMPLOYEE (employee\_id, first\_name, last\_name, hire\_date, job\_title, shop\_id) VALUES

(1, 'George', 'Blake', '2016-12-01', 'Manager', 1),

(2, 'Denise', 'Fredrickson', '2021-03-02', 'Barista', 3),

(3, 'Miguel', 'Rodriguez', '2023-01-01', 'Barista', 2);

INSERT INTO SUPPLIER (supplier\_id, company\_name, country, sales\_contact\_name, email) VALUES

(1, 'Beans R Us', 'United States', 'Alison Kane', 'beansrus@gmail.com'),

(2, 'CreamerCO', 'Germany', 'John Fredrickson', 'creamerco@yahoo.com'),

(3, 'Coffee Pros', 'United Kingdom', 'Paul Grayson', 'coffeepros@gmail.com');

INSERT INTO COFFEE (coffee\_id, shop\_id, supplier\_id, coffee\_name, price\_per\_pound) VALUES

(1, 1, 2, 'Arabica', 1.50),

(2, 3, 1, 'Robusta', 2.98),

(3, 2, 1, 'Liberica', 3.29);

A screenshot of a computer

Description automatically generated

**B3.**

CREATE VIEW EMPLOYEE\_VIEW AS SELECT employee\_id, CONCAT(first\_name, ' ', last\_name) AS employee\_full\_name, hire\_date, job\_title, shop\_id

FROM EMPLOYEE;

A screenshot of a computer

Description automatically generated

**B4.**

CREATE INDEX idx\_coffee\_name ON COFFEE (coffee\_name);

A screenshot of a computer

Description automatically generated

**B5.**

SELECT \*

FROM EMPLOYEE

WHERE employee\_id > 1;

A picture containing text, screenshot, software, computer icon

Description automatically generated

**B6.**

SELECT shop\_name, company\_name, coffee\_name

FROM COFFEE C

INNER JOIN COFFEE\_SHOP CS ON C.shop\_id = CS.shop\_id

INNER JOIN SUPPLIER S ON C.supplier\_id = S.supplier\_id;

A screenshot of a computer

Description automatically generated