SQL Business Analysis Database Project for Blank Street

(Daniella Smith, 2023)

1. Introduction

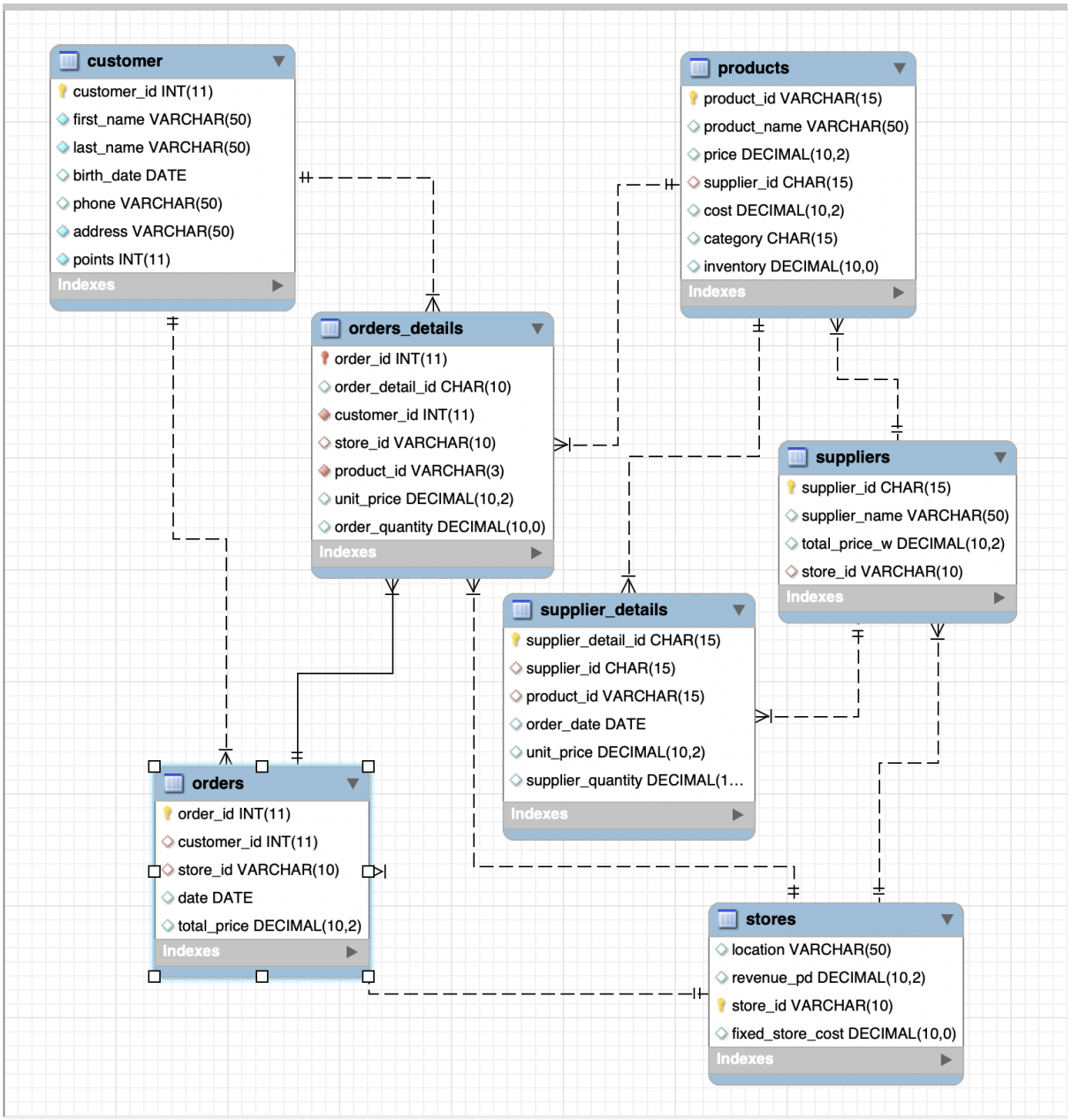
Blank Street Coffee is a US based company and coffee house chain that arrived in the UK in July 2022, and has since expanded across London. Blank Street was founded in New York City, in 2020 by Vinay Menda and Issam Freiha, who previously co-founded VC firm Reshape Ventures. In the three years since they started, Blank Street has established 65 locations, 17 of which are in the UK. In an attempt to out-compete established coffee chains such as Starbucks, Costa and Greggs, Blank Street prioritises local produce, high-tech coffee and aesthetics, in an attempt to scale up the specialty coffee house model.

By having limited-edition drinks, an instagram friendly interior design, and artisanal baked goods, Blank Street is trying to reframe the coffee experience.

With its fast expansion in the UK, Blank Street has to balance its operating costs and monitor revenues to compete with market leaders. I have built a mock Blank Street database with five dimensions : stores, suppliers, products, customers and order information. This database should allow Blank Street to retrieve data for business related queries. For instance, Blank Street can retrieve data from **stores** to see the total revenue for each store, as well as variations in store performance. They can also select data from the **customer** table to determine which customers are highly loyal, and Blank Street could use this data to perform targeted marketing activities. Further, they could examine data from the **supplier_details, supplier** and **stores** tables to calculate the total weekly product bought from a particular supplier for each store, which could be used to enhance operations within stores, and across the chain. With the help of this relational database, Blank Street can enhance its resource allocation, make data driven business decisions, and map out effective marketing strategies

2. Database Design

Figure 1: Database Model



I built 7 tables into the database. The tables are : **customer**, **orders**, **orders_details**, **suppliers**, **suppliers_details**, **stores and products.**

3. Tables Schemes

Customer Table

Figure 2 : customer Table Field View

customer_id	first_name	last_name	birth_date	phone	address	points
1	Daniella	Jackson	2002-08-14	07826509167	7 Nile House	450
2	Julia	Marshall	1997-04-19	32452039853	1 Binch City	390
3	Rachel	Fisher-Quann	2001-08-09	34342345322	10 Internet St	2424
4	Percy	Smith	1993-08-18	02342389532	12 Half-Blood Cl	45
5	Sarah	Hava	1988-02-24	34235342423	33 Podcast St	2333
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 3 : customer Table Setting

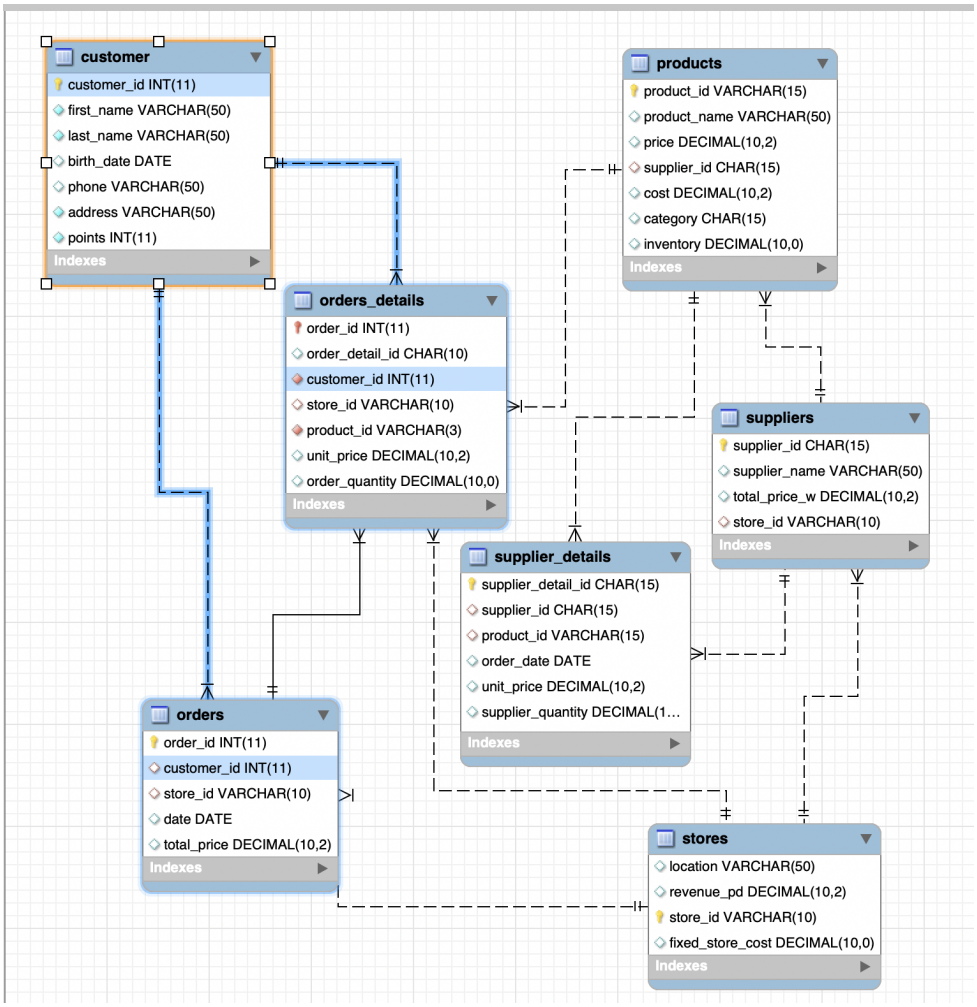
Column	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	G	Default / Expression
customer_id	INT	☑	☑	☐	☐	☐	☐	☑	☐	
first_name	VARCHAR(50)	☐	☑	☐	☐	☐	☐	☐	☐	
last_name	VARCHAR(50)	☐	☑	☐	☐	☐	☐	☐	☐	
birth_date	DATE	☐	☐	☐	☐	☐	☐	☐	☐	NULL
phone	VARCHAR(50)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
address	VARCHAR(50)	☐	☑	☐	☐	☐	☐	☐	☐	
points	INT	☐	☑	☐	☐	☐	☐	☐	☐	'0'
<click to edit>		☐	☐	☐	☐	☐	☐	☐	☐	

I stored information pertaining to specific customers in the **customers** table, assuming this data would be collected via an app or a wifi login process. This table contains a unique customer ID, their first name and last name, their birth date, phone number and address, as well as any points they have accrued via a loyalty program. For customers without a loyalty card, the default number of points is '0'.

For example: Julia Marshall, born in 1997 has a customer ID of 2 and has 390 points on her card.

Primary Key: Customer ID

Figure 4 : customer Table Connections



Orders Table

Figure 5: orders Table Field View

order_id	customer_id	store_id	date	total_price
O1	1	EC2Y	2023-07-07	3.90
O2	2	W1F	2023-07-07	4.10
O3	4	WC2N	2023-07-07	4.20
O4	1	EC2N	2023-07-07	3.90
NULL	NULL	NULL	NULL	NULL

Figure 6: orders Table Setting

Column	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	G	Default / Expressior
order_id	INT	☑	☑	☐	☐	☐	☐	☑	☐	
customer_id	INT	☐	☐	☐	☐	☐	☐	☐	☐	NULL
store_id	VARCHAR(10)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
date	DATE	☐	☐	☐	☐	☐	☐	☐	☐	NULL
total_price	DECIMAL(10,2)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
<click to edit>		☐	☐	☐	☐	☐	☐	☐	☐	

The **orders** table stores information pertaining to customer orders. This table contains a unique order ID, the corresponding customer ID and store ID, as well as the date of the transaction and the total price of the order.

For Example : Order with ID O2 was made on the 7/7/2023 by customer 2, at store W1F, and they spent £4.10.

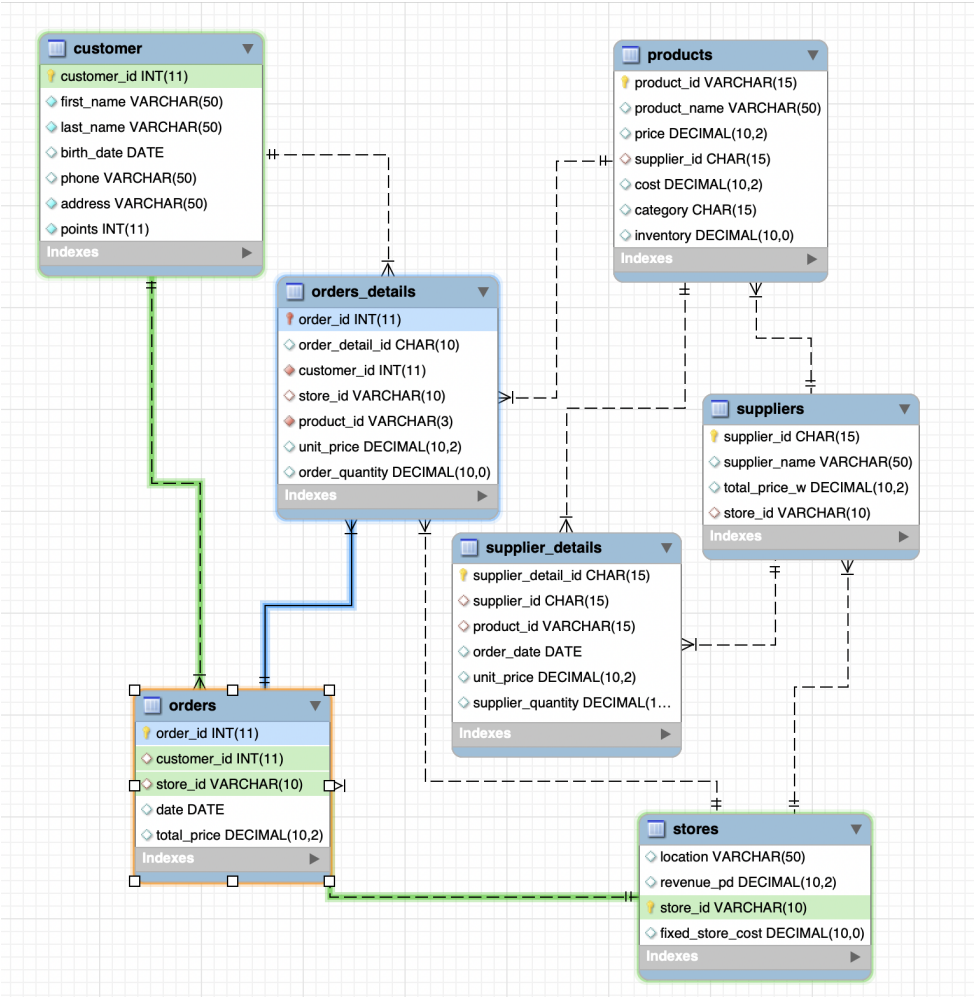
Primary Key : order_id

Foreign Keys:

Customer_id (references **customer** table)

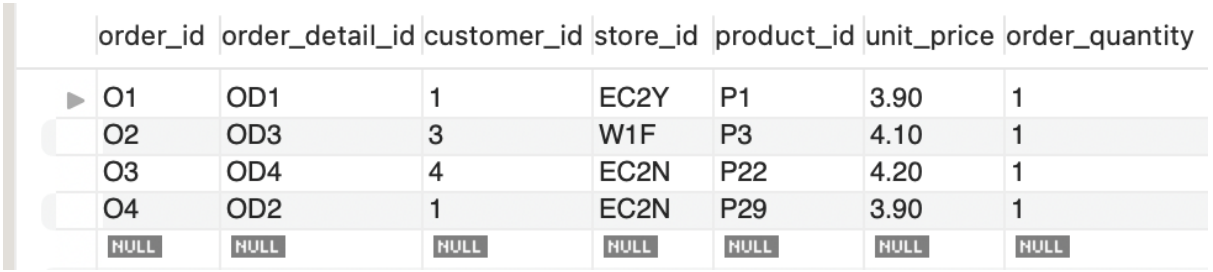
Store_id (references **stores** table)

Figure 7: orders Table Connections



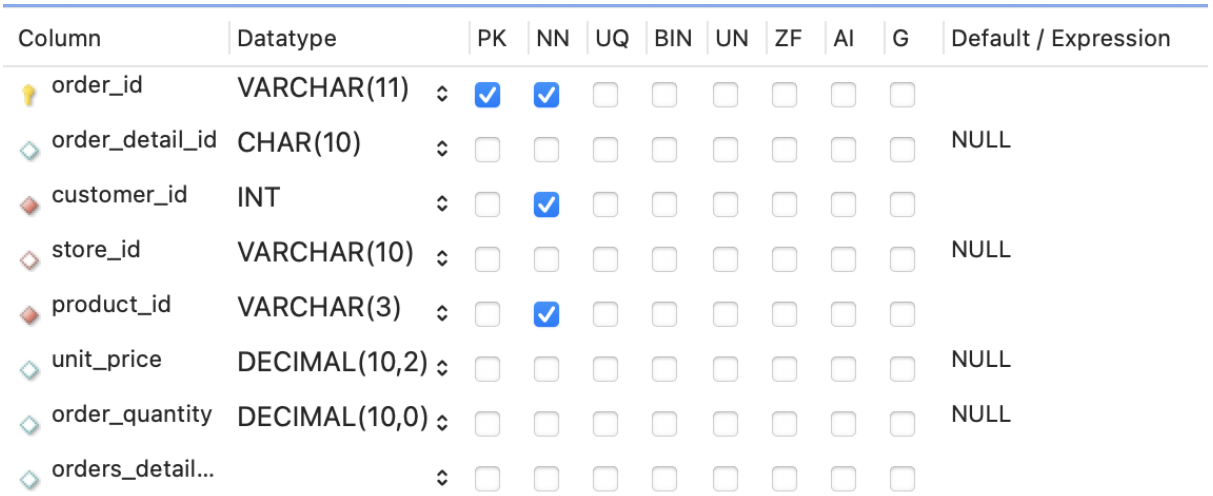
Orders Details Table

Figure 8: orders_details Table Field View



order_id	order_detail_id	customer_id	store_id	product_id	unit_price	order_quantity
O1	OD1	1	EC2Y	P1	3.90	1
O2	OD3	3	W1F	P3	4.10	1
O3	OD4	4	EC2N	P22	4.20	1
O4	OD2	1	EC2N	P29	3.90	1
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 9: orders_details Table Setting



Column	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	G	Default / Expression
order_id	VARCHAR(11)	☑	☑	☐	☐	☐	☐	☐	☐	
order_detail_id	CHAR(10)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
customer_id	INT	☐	☑	☐	☐	☐	☐	☐	☐	
store_id	VARCHAR(10)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
product_id	VARCHAR(3)	☐	☑	☐	☐	☐	☐	☐	☐	
unit_price	DECIMAL(10,2)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
order_quantity	DECIMAL(10,0)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
orders_detail...		☐	☐	☐	☐	☐	☐	☐	☐	

The **orders_details** table stores extra information regarding the orders that customers have made. This table references the order ID, customer ID, store ID and product ID, as well as a unique order detail ID, unit price for the product, and order quantity.

For example: Order O2 for product P3 made by customer 2 in store W1F, and the revenue was £4.10 for 1 unit of the product.

Primary Key : order_detail_id

Foreign Keys:

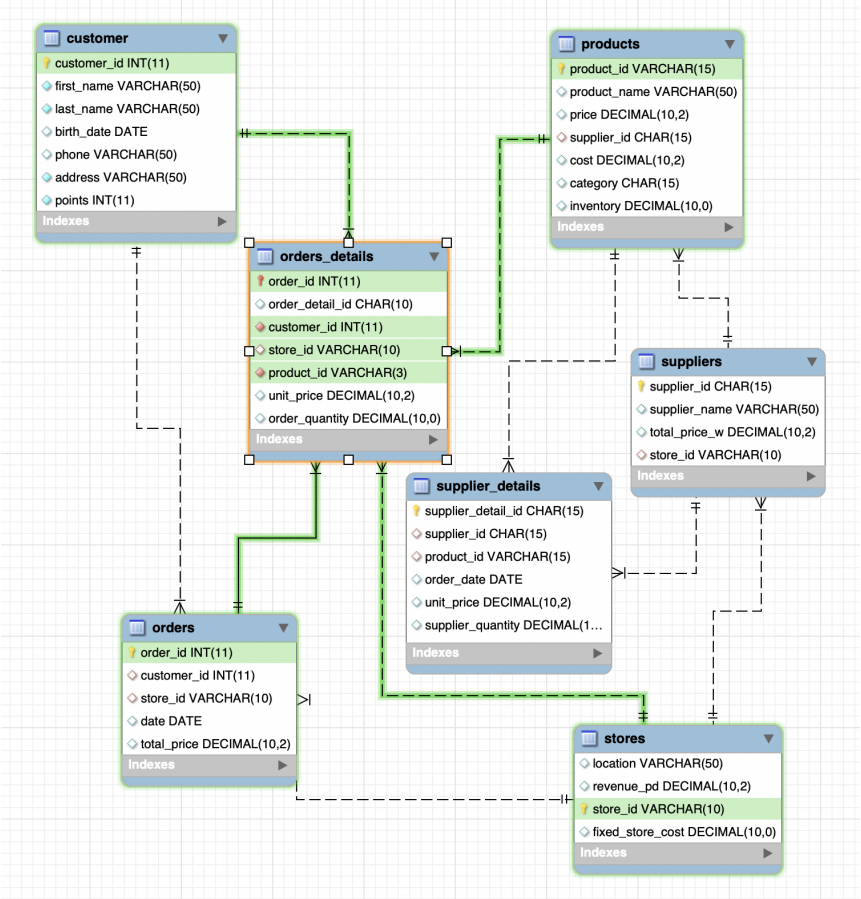
Customer_id (references **customer** table)

Store_id (references **stores** table)

Product_id (references **product** table)

Order_id (references **orders** table**)**

Figure 10: orders_details Table Connections



Suppliers Table

Figure 11: suppliers Table Field View

supplier_id	supplier_name	total_price_w	store_id
S1	Origin Coffee	775.00	EC2Y
S2	Estate Dairy	95.20	W1F
S3	Monin Syrups	79.00	EC2N
S5	PerfectTed	16.50	EC2Y
NULL	NULL	NULL	NULL

Figure 12: suppliers Table Field View

Column	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	G	Default / Expression
supplier_id	CHAR(15)	☑	☑	☐	☐	☐	☐	☐	☐	
supplier_name	VARCHAR(50)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
total_price_w	DECIMAL(10,2)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
store_id	VARCHAR(10)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
<click to edit>		☐	☐	☐	☐	☐	☐	☐	☐	

The **suppliers** table stores information about the orders made to suppliers of wholesale items, for example, the coffee beans required for the coffee shop. This table contains a unique supplier ID, alongside the supplier's name, the total price of the items ordered from the supplier per week, and references the store ID.

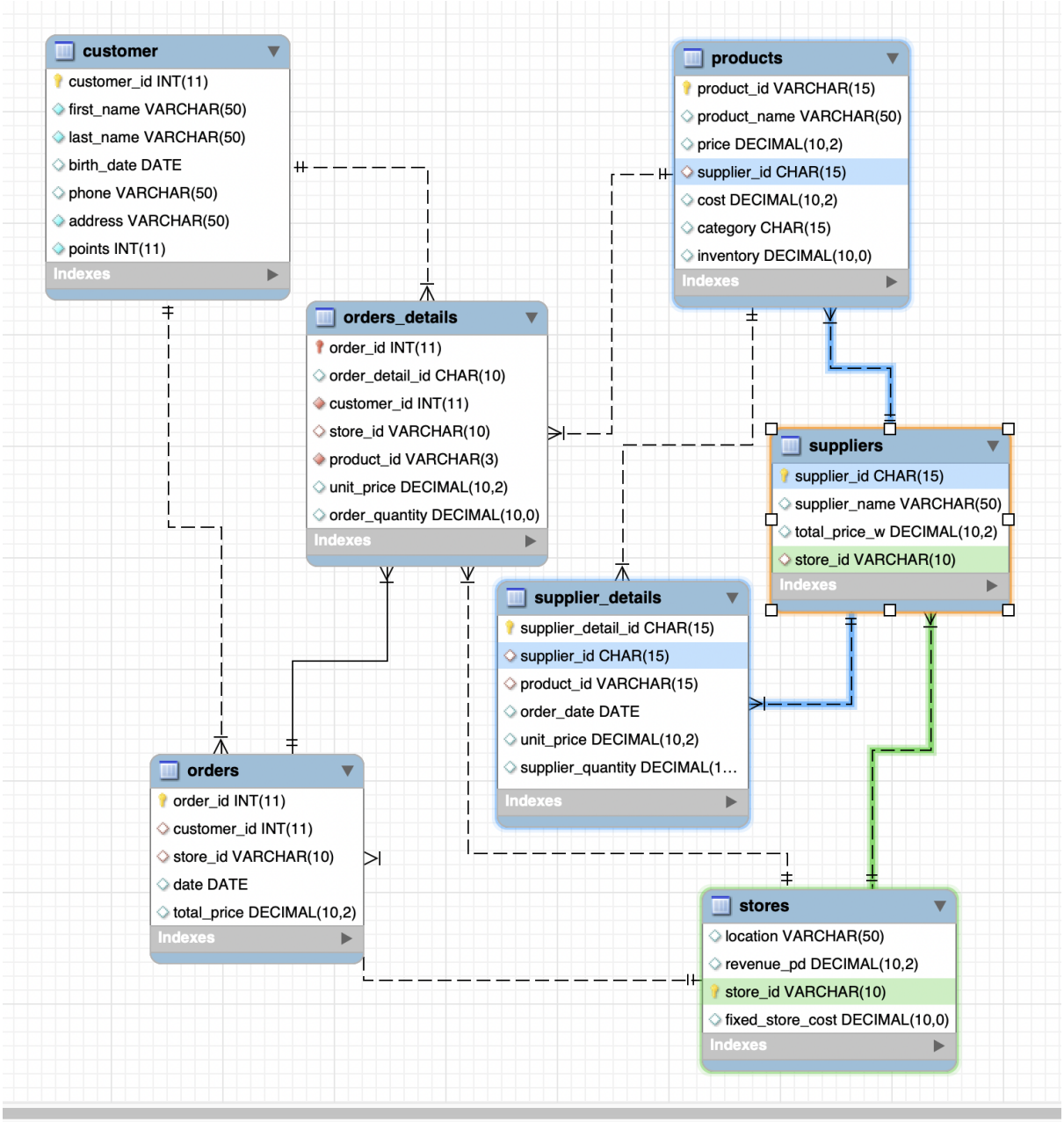
For example : The store with id 'EC2Y' ordered £775 of product from Origin Coffee, which has a supplier ID of 'S1'.

Primary Key : supplier_id

Foreign Keys:

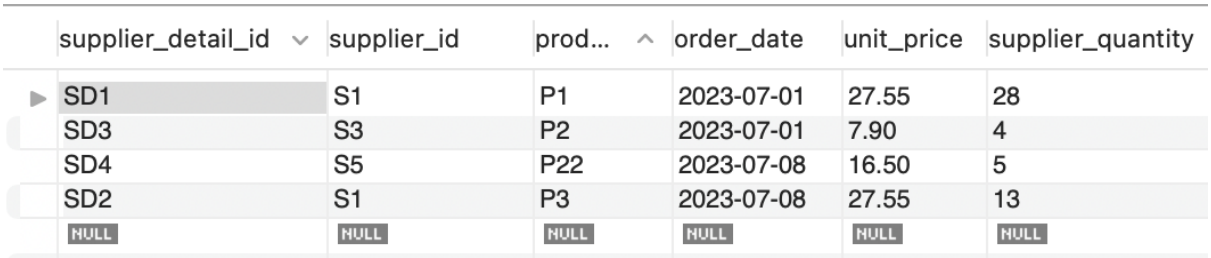
Store_id (references **stores** table)

Figure 13: suppliers Table Connections



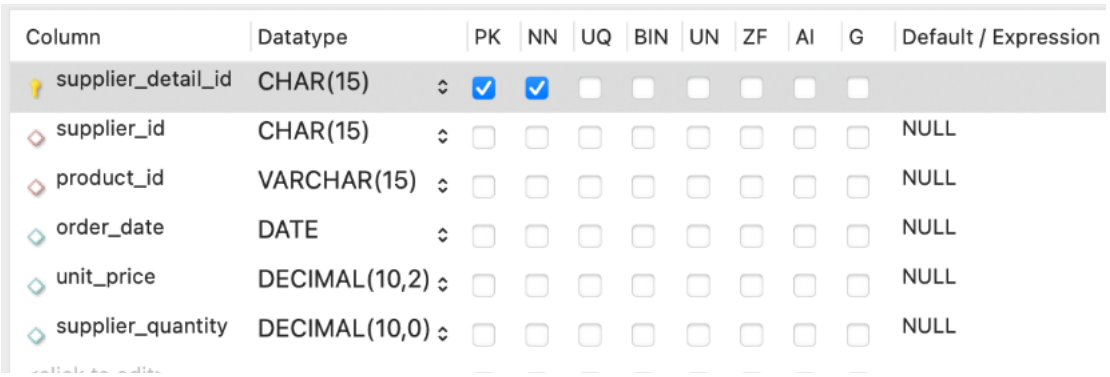
Supplier Details Table

Figure 14: supplier_details Table Field View



supplier_detail_id	supplier_id	prod...	order_date	unit_price	supplier_quantity
SD1	S1	P1	2023-07-01	27.55	28
SD3	S3	P2	2023-07-01	7.90	4
SD4	S5	P22	2023-07-08	16.50	5
SD2	S1	P3	2023-07-08	27.55	13
NULL	NULL	NULL	NULL	NULL	NULL

Figure 15: supplier_details Table Setting



Column	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	G	Default / Expression
supplier_detail_id	CHAR(15)	☑	☑	☐	☐	☐	☐	☐	☐	
supplier_id	CHAR(15)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
product_id	VARCHAR(15)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
order_date	DATE	☐	☐	☐	☐	☐	☐	☐	☐	NULL
unit_price	DECIMAL(10,2)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
supplier_quantity	DECIMAL(10,0)	☐	☐	☐	☐	☐	☐	☐	☐	NULL

The **supplier_details** table has additional information about the orders made from suppliers. This table stores a unique supplier_detail_id, the date of the transaction, the per-unit price of the goods purchased, and the quantity of units purchased. It also references the supplier id, as well as the id of the product for which the goods are being purchased.

For example: SD3 informs us that on the 1st of July 2023, 4 units, priced at £7.90, were purchased from the supplier with ID S3, and that this ingredient is necessary for the product with ID P1.

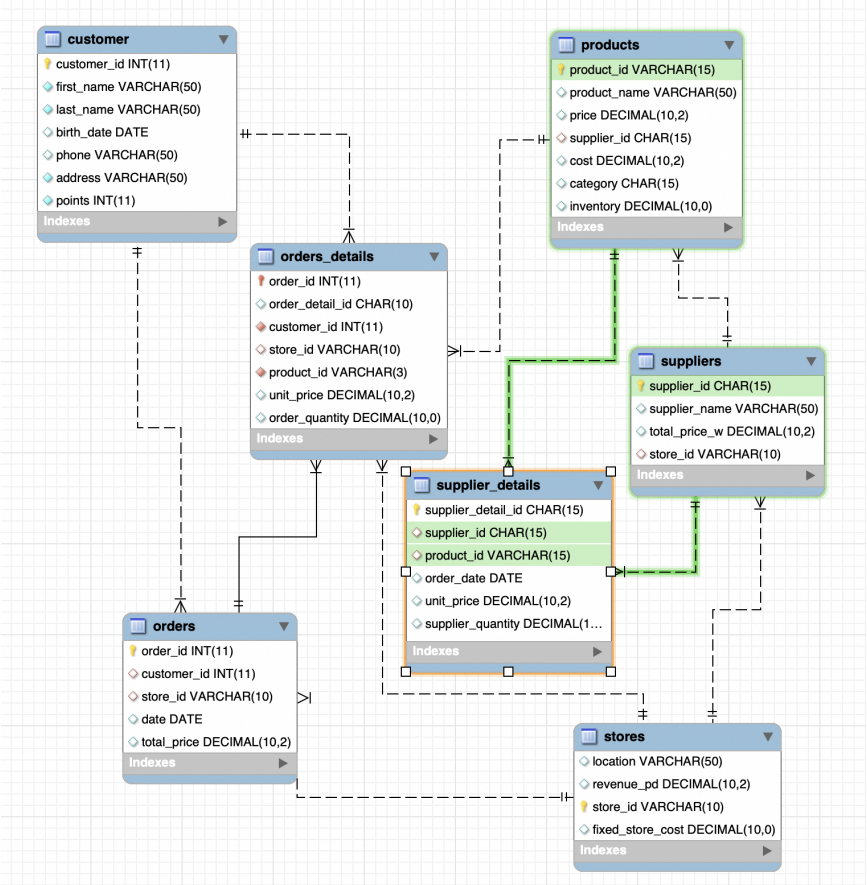
Primary Key : supplier_details_id

Foreign Keys:

supplier_id (references **suppliers** table)

product_id (references **product** table)

Figure 16: supplier_details Table Connections



Stores Table

Figure 17: stores Table Field View

location	revenue_pd	store_id	fixed_store_cost
city	7000.00	EC2N	13000
moorgate	4500.00	EC2Y	9500
notting hill	6500.00	W11	8900
strand	6000.00	WC2N	10000
westminster	6700.00	W1F	12000
NULL	NULL	NULL	NULL

Figure 18: stores Table Setting

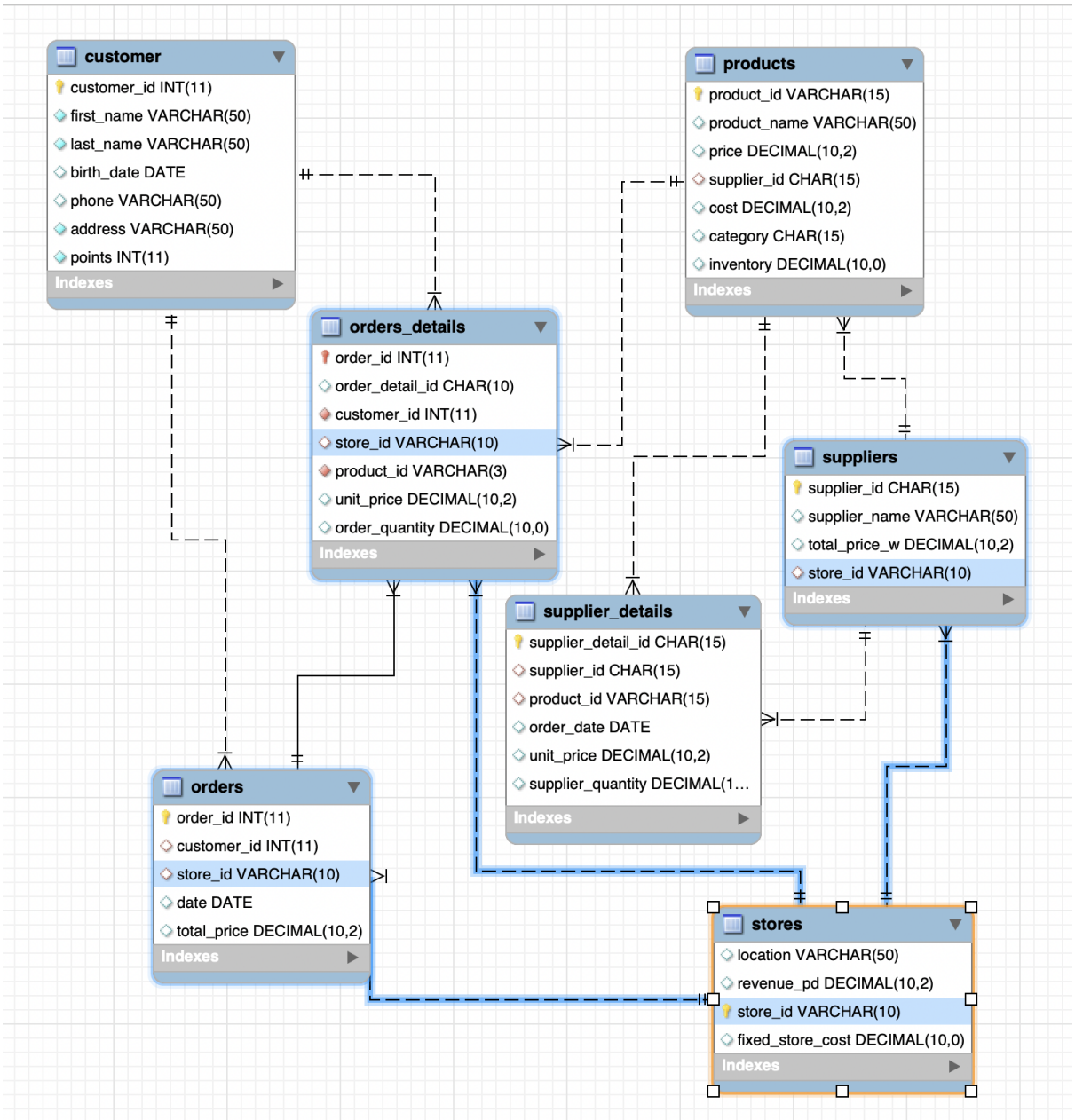
Column	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	G	Default / Expression
location	VARCHAR(50)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
revenue_pd	DECIMAL(10,2)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
store_id	VARCHAR(10)	☑	☑	☐	☐	☐	☐	☐	☐	
fixed_store_c...	DECIMAL(10,0)	☐	☐	☐	☐	☐	☐	☐	☐	NULL

I stored information about the stores that Blank Street has established in this table, with the goal of having an index of all the locations. As Blank Street continues to expand, having the details of the operations of each store in one database could ensure that variations in store performance are noted, and fluctuations across stores in different locations can be analysed. The store_id is the same as the store's postcode. The table contains the location's name, the store's average daily revenue (which should be revised every month), and the fixed monthly rent of the store.

For example: The store in moorgate, which has an average daily revenue of £4500 has a store_id of 'EC2Y' and has a fixed monthly rent of £9500.

Primary Key: store_id

Figure 19: stores Table Connections



Products Table

Figure 20: products Table Field View

product_id	product_name	price	supplier_id	cost	category	inventory
P1	original cold brew	3.90	S1	0.72	cold brew	400
P2	shaken brown sugar cold brew	4.70	S3	0.94	cold brew	250
P22	iced blueberry matcha	4.20	S5	0.84	iced coffee	100
P29	flat white	3.90	S1	0.78	hot coffee	500
P3	cortado cold brew	4.10	S1	0.82	cold brew	300
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 21: products Table Setting

Column	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	G	Default / Expression
product_id	VARCHAR(15)	☑	☑	☐	☐	☐	☐	☐	☐	
product_name	VARCHAR(50)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
price	DECIMAL(10,2)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
supplier_id	CHAR(15)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
cost	DECIMAL(10,2)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
category	CHAR(15)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
inventory	DECIMAL(10,0)	☐	☐	☐	☐	☐	☐	☐	☐	NULL

I stored information about the products that Blank Street offers in this table, with the goal of having a dynamic index of all the items on offer. Theoretically, as seasonal items are introduced and retracted, this database could serve as a record of changes in price, product popularity, and changes in suppliers. The table contains a unique product ID, as well as the product's name, price it is sold for, cost, product category, as well as the supplier ID, and an inventory of how much of a product each store should have.

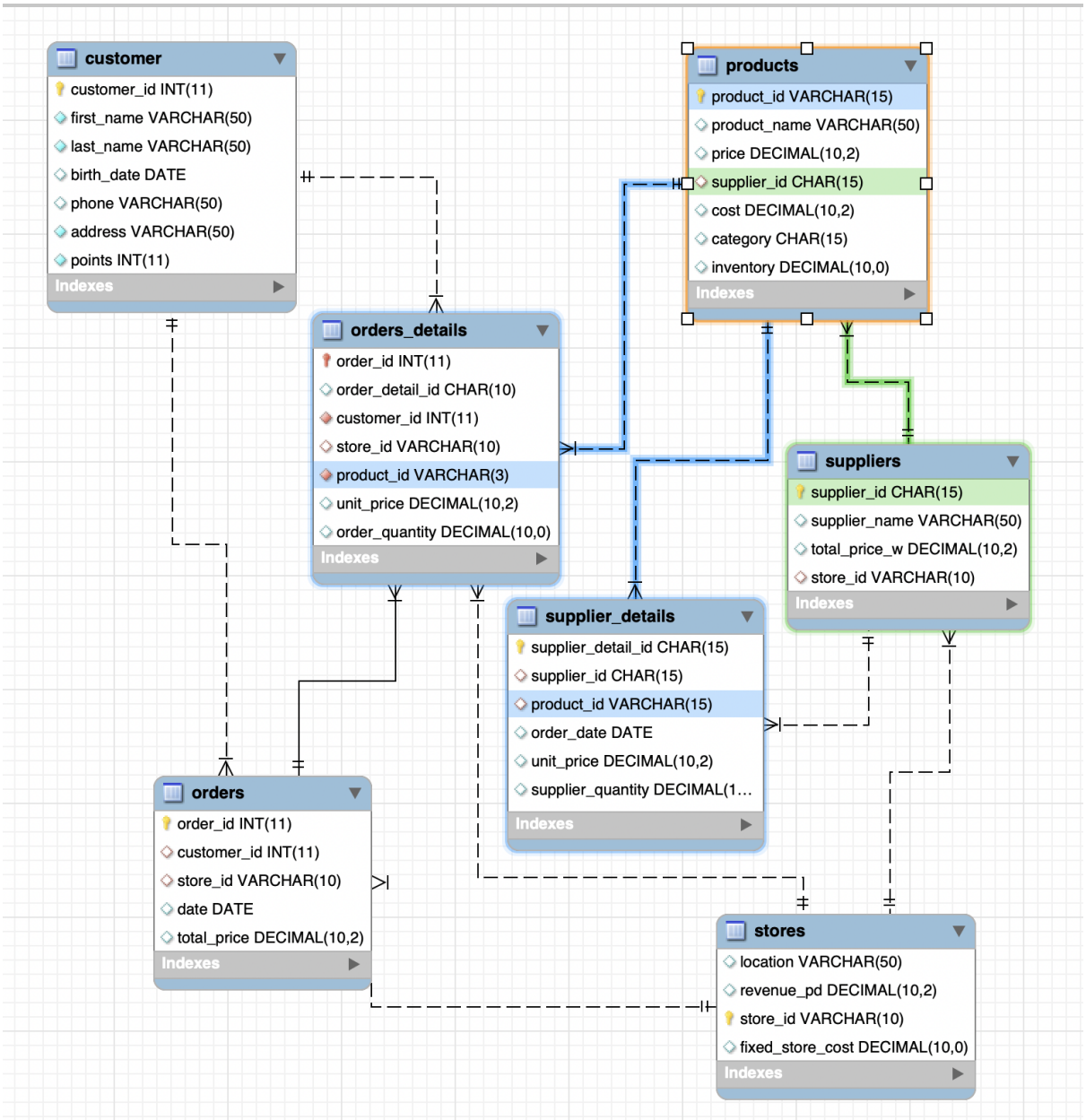
For example: P1 refers to Original Cold Brew, which is sold for £3.90 per unit, costs £0.72 to produce and is supplied by supplier S1. Is is in the 'cold brew' category, and each store should start the week with 400 in stock.

Primary Key: product_id

Foreign Key:

Supplier_id (references **suppliers** table)

Figure 22 : products Table Connections



4. Queries

Store Performance Query

```
8 •     SELECT *
9       FROM stores
0       ORDER BY revenue_pd desc;
1
```

28:200

sult Grid Filter Rows: Search

location	revenue_pd	store_id	fixed_store_cost
city	7000.00	EC2N	13000
westminster	6700.00	W1F	12000
notting hill	6500.00	W11	8900
strand	6000.00	WC2N	10000
moorgate	4500.00	EC2Y	9500
NULL	NULL	NULL	NULL

Management wanted to rank the stores by average daily revenue, to find out which store was most profitable. This query puts store revenue in descending order, revealing 'city' to be the most successful store, and 'moorgate' to be the least. Further as it calls all the columns, revenue can be compared to the fixed store cost (rent).
Management can use information regarding the relationship between store location and revenue to seek opportunities in high performing locations, and maintain a high ROI when it comes to fixed store costs.

Union : Customer Loyalty

```
211 •    SELECT customer_id,first_name,points,'bronze' AS type
212     FROM customer
213     WHERE points < 1000
214     UNION
215     SELECT customer_id, first_name, points,'silver' AS type
216     FROM customer
217     WHERE points BETWEEN 1000 AND 2000
218     UNION
219     SELECT customer_id, first_name, points,'gold' AS type
220     FROM customer
221     WHERE points >= 2000
222     ORDER BY points DESC;
```

100% 22:222

Result Grid Filter Rows: Search Export:

customer_id	first_name	points	type
3	Rachel	2424	gold
5	Sarah	2333	gold
8	Isabela	1599	silver
7	Pablo	1002	silver
10	Ava	790	bronze
1	Daniella	420	bronze
2	Julia	390	bronze
6	Peter	120	bronze
4	Percy	45	bronze
9	Alma	25	bronze

This query introduces an additional column to the **customer** table, sorting them into 'types' according to the amount of points they have accrued.
A bronze customer is defined by one with less than 1000 points, a silver one has between 1000 and 2000 points, and a gold customer has over 2000 points.
Assuming a loyalty program where customers can gain a set amount points per beverage purchased, and can spend points after certain statuses or points have been acquired, management can adjust promotions for different groups of customers.

For example, management could direct promotions trying to ensure customer loyalty and acquisition for people typed as 'bronze', and could have a different set of promotions trying to ensure retention and satisfaction for customers marked as 'silver' or gold.

Inner Join : Connecting order details to product information

```
243 •  SELECT order_id, o_d.product_id, product_name, unit_price, category, customer_id
244    FROM orders_details as o_d
245    JOIN products as p
246          ON o_d.product_id = p.product_id;
```

100% 36:246

Result Grid Filter Rows: Search Export:

order_id	product_id	product_name	unit_price	category	customer_id
O1	P1	original cold brew	3.90	cold brew	1
O2	P3	cortado cold brew	4.10	cold brew	3
O3	P22	iced blueberry matcha	4.20	iced coffee	4
O4	P29	flat white	3.90	hot coffee	1

This query is designed to join the orders_details table and the products table, so for each order in the system, we return the order_id, the product_id, the product's name, the unit price, the category of product, and the customer ID.

With a large database, management could examine recurring customer_ids to see which are the most popular products among loyal customers, and which are the most popular products amongst new clientele. Additionally, management could look at different orders to see which items are often ordered together, for example, if a customer frequently orders a baked good with a hot drink. In this way, management can plan promotions, can adjust prices to enhance profit, and can have easy access to essential information.

Outer Join

```
248 •  SELECT st.store_id, st.location,sd.order_date,sd.product_id,s.total_price_w
249    FROM supplier_details sd
250    LEFT JOIN suppliers s
251        ON sd.supplier_id = s.supplier_id
252    LEFT JOIN stores as st
253        ON s.store_id = st.store_id;
```

100% 30:253

Result Grid Filter Rows: Search Export:

store_id	location	order_date	product_id	total_price_w
EC2Y	moorgate	2023-07-01	P1	775.00
EC2Y	moorgate	2023-07-08	P3	775.00
EC2N	city	2023-07-01	P2	79.00
EC2Y	moorgate	2023-07-08	P22	16.50

This query is designed to perform an outer join, where the supplier_details table is left joined onto the suppliers and stores tables.
For every supplier transaction, we call the store_id, location, order date, the name of the product to which the order corresponds, and total weekly outgoings to the supplier in relation to that product.

Management can use this to see which stores are making the most orders from suppliers, and which products are requiring the most frequent supplier deliveries. If there are discrepancies between stores in how often a supplier has to send a product, this may indicate that a certain store is selling a particular product at a higher rate than expected, or conversely, that there is an issue with stock management at that store.

Furthermore, this outer join can be used to calculate the total weekly product bought from a particular supplier across all the stores in the company; if a higher than anticipated amount of product is being supplied, data from this table could be used to leverage a more advantageous wholesale discount negotiation.

Filtered Data With WHERE Query

```
255 •  SELECT o.total_price,s.revenue_pd,s.store_id
256    FROM orders as o,stores as s
257    WHERE o.store_id = s.store_id
258    ORDER BY s.revenue_pd DESC
259    LIMIT 1;
260
```

100% 9:259

This query is designed to call the order with the highest price as well as the store with the highest average daily revenue into the table. Management could use this to alter their product offering according to consumer trends, for example, if the stores with the highest revenue tend to have smaller highest priced orders, that may indicate that store performance is dictated by their ability to sell low cost items, instead of high cost items.

Creating A View : Low Performing Customers

```
CREATE
ALGORITHM = UNDEFINED
DEFINER = `root`@`localhost`
SQL SECURITY DEFINER
VIEW`Blank_Street`.`low_performance_customer` AS
SELECT
`Blank_Street`.`customer`.`customer_id` AS `Customer_ID`,
`Blank_Street`.`customer`.`first_name` AS `First_Name`,
`Blank_Street`.`customer`.`last_name` AS `Last_Name`,
`Blank_Street`.`customer`.`points` AS `Points`
FROM
`Blank_Street`.`customer`
WHERE
(`Blank_Street`.`customer`.`points` <100)
ORDER BY
`Blank_Street`.`customer`.`Points`;

SELECT * FROM Blank_Street.low_performance_customer;
```

100% 54:18

Result Grid Filter Rows: Search Export:

Customer_ID	First_Name	Last_Name	Points
9	Alma	Norgaard	25
4	Percy	Smith	45

This query is designed to create a view that allows management to see 'low performing customers', or customers with less than 100 points. Assuming that a customer gains points by purchasing items, with points perhaps having a time limit (e.g. 5 points lost per week since last purchase), it may be useful to see which customers have very low points, so management can direct marketing activities towards them.