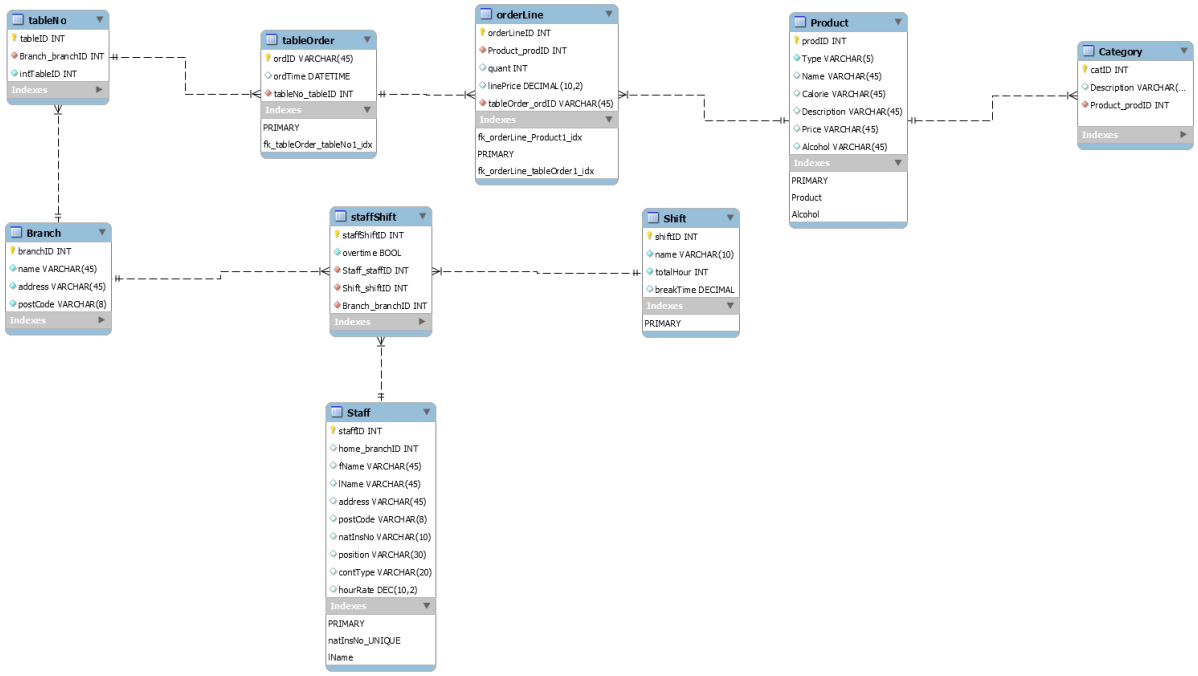
Q4 – Database Monitoring and Optimization

Database monitoring and optimisation is an essential part of ongoing maintenance and general upkeep. As the database becomes more in demand in terms of the number of IOPS previously small-time delays may become more apparent and their effects may become more apparent. With regards to CC, the more customers and branches it attracts/opens will increase the number of transactions being processed which could lead to detrimental effects if the database is not properly maintained.

Database Design Re-Evaluation

The database structure defined in submission 1 has been modified in ways that will hopefully improve database efficiency as well as allowing for queries to become simpler (in terms of lessening the number of required joins). The new EERD is show below:



The main changes include denormalization of both the product and staff tables. The staff tables before were normalized into either management staff or floor staff, products were also normalized into either food or drink stuffs. The first step of optimization is re-evaluating the existing database design, in cases such as this is may be necessary to redesign certain parts of a DB to improve efficiency as well as improve query speeds. Having larger tables instead of smaller ones means that the number of indexes required lessens. This could then lead to a decrease in space required within storage facilities which could in turn reduce running costs.

Attributes have also been removed from the original EERD, this is because through re-evaluation is has become apparent that particular attributes were irrelevant and therefore not needed. Again, this helps with database efficiency as well as reducing overall space occupied by the database.

DB Optimisation Software

As the recommendation for CC's database was an AWS instance of MySQL another tool can be quickly installed on the virtual machine to handle automated optimisation and query optimisation as and when needed.

DB Optimizer

Idera's 'DB Optimizer' can be implemented server/cloud side and will run concurrently with the live database. As and when transactions are carried out the software measures and records particular results such as overall time taken and the relationships that the database must create/utilise when running transactions.

DB Optimizer's GUI implements features such as a 'time' graph showing when the highest delays occur and the transactions that run during these times, through analysis of repeated transactions suggestions for code modifications can be made. These modifications are explained before and commits are carried out, suggestions are viewable in either pure code formats, or as query trees to help define potential new and useful table relationships.

Before changes are made to a database scheme, administrators can also utilise Optimizer's simulation tool which bases results on data gathered through use of the current database schema as opposed to running a direct duplicate of the database purely for testing purposes, not only would this cause more of a time and productivity delay, it would also temporarily double the database size which may not be possible on servers with limited data storage pools therefore making the simulation elements of Optimizer particularly useful when limited resource situations occur.

Periodic Data Review

Periodically the database should be reviewed in terms of how the code is written but more importantly though use of surveys and shadowing methods.

Shadowing can be used to aid search queries, 'shadowers' would be able to see what employees most commonly search for and how their searches are started. This can be undertook with a view to reallocate primary keys to attributes that are more commonly searched for than the PK's initially set during database creation. In context, individuals can be sent to various branches to spend time (during peak times) to analyse what it is employees most regularly search for and just how they search when using a POS system. Whilst not the most technical approach, it can be more beneficial to see how the main user class use the database in their day-to-day work.

Through code analysis and data analysis, it will be possible to see which attributes are being under-utilised and those which are nearing peak utilisation. Attributes (such as customer first name) with defined lengths can be altered at any time. If it became apparent that the maximum field length is being reached more frequently than first thought, the attribute can then be lengthened and if not then vice-versa. Similarly, fields that contain information such as date can also be altered to display a shorter date format. By shortening or lengthening various attributes database costings will change and could become lesser. Code analysis along with the use of optimisation suites (such as those previously mentioned) will allow for administrators to view current code and transaction logs in order to determine areas of inefficiencies as well as areas where the code is physically cluttered. By reorganising the cluttered code this may not affect overall performance but may impact future maintained speed as code may be easier to decipher.

Execution planning

When transactions are run (either during up for downtimes) it is important to determine the order in which transactions and accompanying processes are run. Within particular databases and businesses, it would be important for a monetary transaction to be carried out before a stock record is altered, this would be so that the business can ensure money is taken and payment has been received. Stock records can be altered later by compiling the days transactions into a single report rather than making multiple smaller transactions throughout the day.

Execution planning plays a key role in ensuring continued database performance as it prevents unnecessary occurrences of transactions that would otherwise reduce overall speed of the database. Within CC as previously mentioned within the database assumptions, stock records are accounted for during downtime. This therefore means that the likelihood of running stock record modification transactions concurrently with orders should not be possible. The transaction should remain as a stored procedure completed at a specific time set by the system administrator. Reports in general should always be run during system downtime, in the case of CC this will be during closing hours where stock records can then be consolidated without new orders being processed, reordering reports can then also be generated and sent to the relevant contacts to arrange product deliveries, as mentioned in the assumptions, this database does not handle deliveries, only products to be re-ordered are generated.

Query Trees and Relational Algebra

The use of query trees and relational algebra allow current and future administrators to determine just how complex queries may work. Through critical analysis using these optimisation methods administrators and designers would then be able to use the aforementioned techniques to reorder queries differently or determine where improvements could be made. For example, removing joins that may not be necessary in hindsight.

Modification of joins may also become necessary once QT and RA analysis has been carried out. Different types of joins such as ‘Full Outer’ are very slow and so can be modified to provide faster return times. Modifications can include complete redesigns of the code and/or utilisation of different join types. Through us of DB optimisers the code and easily be redesigned to be less demanding on the servers and therefore faster.

The key aspects of relational algebra are as follows (Ghemri, 2019):

- It provides a formal foundation for all relational database models
- It can be used as a basis to implement and optimise existing queries which are of a higher level of importance than others
- Some RA concepts are already built into SQL languages and so translation is simple

Query trees provide a more visual representation that just RA alone. You are able to see that steps taken during the execution of a query. Through use of similar notation trees can then be converted into RA more effectively. With optimisation in mind the trees allow for clear representation of how queries are laid out, therefore it would become apparent quicker as to what can be modified therefore potentially allowing for a faster scheme to be built in a lesser amount of time.

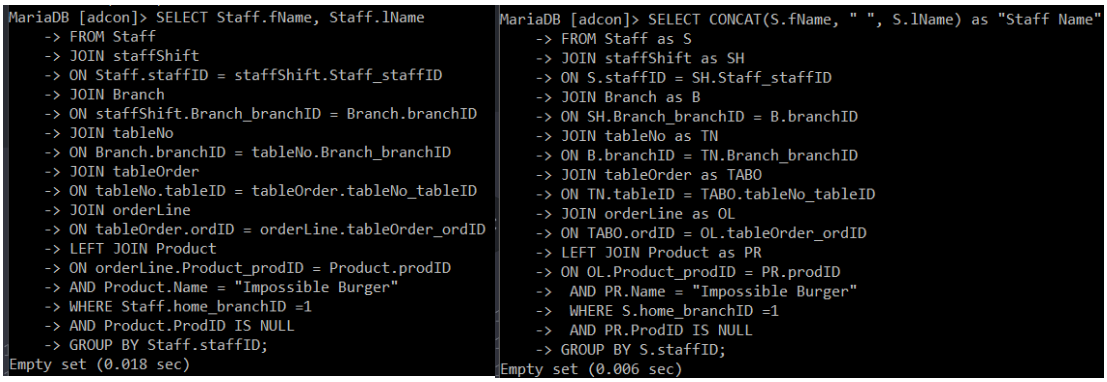
By using user feedback when determining queries to optimise the RA and trees will aid in determining the steps in a query that may be the most detrimental to execution time. Exact use cases for these tools are difficult to define at this moment in time, they will become useful when database monitoring commences once the database goes live within the context of CC pubs.

Indexing

Following on from the aforementioned data review, indexes should also be reviewed and revised if necessary. By using a mission critical database and including multiple indexes where may not be necessary can affect return speed and transaction times in a negative way. As the DBA learns how users search through the database through use of optimisation software or DB logs they can then draw up which data is 'touched' when running transactions or queries. This is an effective way to determine if the current index setups are still valid and current, if data that is not indexed is being referenced more than that currently indexed data then this suggests that indexes must be revised. By removing unnecessary indexes transaction speeds can be reduced as less stress is then being put onto the database which can then allow for more transactions to be carried out. Removing indexes also reduces the overall size of the database, reiterating a previous statement, this would then save money in terms of data storage allocations from AWS. The money saved could then be put into development of other database systems such as NoSQL which in the case of CC could provide more detailed marketing insights based on a wider range of data.

Code Optimisation Example

Below is detailed an identical query which is designed to return an empty set of data. The first query includes no formatting whatsoever and does not use any abbreviations for table names. By simply implementing these features, query speeds can be reduced. If the query is executed across all 60 branches 20 times a day the time differences quickly add up.



```
MariaDB [adcon]> SELECT Staff.fName, Staff.lName
    -> FROM Staff
    -> JOIN staffShift
    -> ON Staff.staffID = staffShift.Staff_staffID
    -> JOIN Branch
    -> ON staffShift.Branch_branchID = Branch.branchID
    -> JOIN tableNo
    -> ON Branch.branchID = tableNo.Branch_branchID
    -> JOIN tableOrder
    -> ON tableNo.tableID = tableOrder.tableNo_tableID
    -> JOIN orderLine
    -> ON tableOrder.ordID = orderLine.tableOrder_ordID
    -> LEFT JOIN Product
    -> ON orderLine.Product_prodID = Product.prodID
    -> AND Product.Name = "Impossible Burger"
    -> WHERE Staff.home_branchID =1
    -> AND Product.ProdID IS NULL
    -> GROUP BY Staff.staffID;
Empty set (0.018 sec)
```

```
MariaDB [adcon]> SELECT CONCAT(S.fName, " ", S.lName) as "Staff Name"
    -> FROM Staff as S
    -> JOIN staffShift as SH
    -> ON S.staffID = SH.Staff_staffID
    -> JOIN Branch as B
    -> ON SH.Branch_branchID = B.branchID
    -> JOIN tableNo as TN
    -> ON B.branchID = TN.Branch_branchID
    -> JOIN tableOrder as TABO
    -> ON TN.tableID = TABO.tableNo_tableID
    -> JOIN orderLine as OL
    -> ON TABO.ordID = OL.tableOrder_ordID
    -> LEFT JOIN Product as PR
    -> ON OL.Product_prodID = PR.prodID
    ->  AND PR.Name = "Impossible Burger"
    ->  WHERE S.home_branchID =1
    ->  AND PR.ProdID IS NULL
    -> GROUP BY S.staffID;
Empty set (0.006 sec)
```

The left query is designated as the slower variant whereas the right is the faster query through use of the aforementioned features. Using the figures previously mentioned, the left query would complete in 21.6 seconds whereas the right would complete in 7.2 seconds. As optimisation is required across all queries the total times would add up quickly to significant amount and therefore significant saving.

Q5 – Option C – Mobile Database

Introduction

Crazy Cat pubs decision to integrate tablets into their POS systems represents a step forward in terms of customer service allowing customers to order products from a table as opposed to the traditional method whereby customers would wait at the bar for service. Tablets used at the bar will allow for staff to be more mobile as opposed to returning to the same POS to process an order, during peak hours this may be far more difficult than during lull periods.

Additional Assumptions

Due to the additional hardware and database modifications required there are additional assumptions that must be enforced to ensure a smooth transition is applied.

Tablets are able to run for long periods of time without needing to charge. Batteries must be sufficient to run a web client for extended periods of time.

Tablets are bundled with rugged cases to protect the screen and body against damage whether this be accidental or malicious.

Tablets are remotely monitored and managed primarily to ensure updates are smoothly carried out and security lock-outs and device tracking can be carried out If the need arises.

Tablets must run a recent version of Android with a corporate launcher installed to limit user actions. The database would be accessed through a web client reducing the need to have high end devices thus reducing initial expenditure.

As tablets will be used within an enterprise environment, additional software is required to help enforce permissions and application installs. BlueFetch will be the system of choice in this instance and so additional costings will be incurred (later explained).

Payments are carried out on other devices carried by staff members; the tablets do not oversee transactions and only handle product orders.

Database Modifications

An additional table would need to be implemented within the database to handle tablet ownership to each branch. This would handle tablet identifiers and software versions installed.

The structure of this new table would be as follows:

Table Name: MobilePOS

Attribute Name	Description	Data Type	Special Properties
TabID	Tablet Identifier	INT	Primary Key
TabMAC	Tablet Mac Address	VARCHAR(11)	Unique, Not Null
SoftVers	Software Version (Android)	Dec	
Branch_BranchID	Branch Allocation	INT	Foreign Key

Order tables would also need to be modified to add a device ID as to determine elements such as the following:

- Tablet utilisation
- Most popular tables
- Average number of orders placed per hour (can then be used to introduce 'Happy Hour' to increase customer throughput)

At current, the ordering table 'tableOrder' is in the current format:

Attribute Name	Description	Data Type	Special Properties
orderLine_orderID	Order Identifier	INT	Primary Key
orderLine_Food_foodID	Food Identifier	INT	Unique, Not Null
orderLine_Drinks_drinkID	Drink Identifier	INT	
tableNo_tableID	Table Identifier	INT	Foreign Key, Not Null
tableNo_Branch_BranchID	Branch ID	INT	
ordTime	Time of transaction	Time	

Another attribute would need to be added to the 'tableOrder' table with a foreign key of the TabID. This would allow all transactions to be traced back to the tablet used to carry out the transaction, therefore the staff member can also be determined.

Security and Mitigation

As tablets are portable devices there are several different risks that can be introduced. However, these risks are met with mitigation methods that can be implemented in a variety of different ways. Risks present may lead to elements of GDPR and DPA being breached. Particularly those concerning customer orders and employee records.

Risk: Tablet is Lost

If a tablet is misplaced the information present on the device may cause vulnerabilities such as data leaks. However, the software present on the tablets and the fact Android provides device security built-in which allows remote secure deletion or remote disabling of the device. Provided the devices utilise a 4G data link (if available) tracking will be possible even if the device is out of range. In terms of data being present on the device, the amount of data is very insignificant. AWS hosts all data (as mentioned in Q3) and would be accessed through a web portal as opposed to holding a local variant of the database.

Risk: Tablet is Damaged

If the tablet is damaged (beyond repair) a service agreement between tablet providers and CC will depict that a replacement device would be provided within good time. This therefore would mean little downtime if a tablet is damaged thus lessening the impact on service times for customers.

Risk: Malicious Customer Uses Tablet

The risk of malicious customers using tablets could cause problems such as orders being sent to the incorrect tables as well as orders being put through without payment. This risk can be mitigated by enforcing policies such as 2 step verification whereby users would be requested to enter the branch number followed by scanning their NFC ID tag. Screen timeouts can also be used to lock a device after a short amount of time to limit potential unsecured exposure.

Encryption

In order for tablet communications with access points and ultimately the database secure, the access points should be implemented with WPA-2 Enterprise encryption. This type of encryption

provides a secure key and secure communications once setup correctly, malicious attempts to access the network are very unlikely due to the complexity of passphrases. The key itself should only be stored by system administrators or network integrators to prevent the code being handed out by malicious employees and therefore into the public domain. This helps ensure network security as well as bandwidth availability.

Tablet Software

As devices will be deployed within an enterprise environment, device management may be difficult due to the sheer number of devices that could be deployed. MDM (multi device management) software allows administrators to manage all devices in one go as opposed to altering each device one-by-one. MDM tools such as Bluefetch allow for a basic GUI to be displayed allowing access to only a few applications which in the case of CC would be the ordering web portal. Bluefetch handles tablet login through use of an NFC plugin, this is similar to the POS login system discussed in Q3 as an NFC tag can be held to the back of the device to allow a user to login quickly.

Figure 1 (Bluefetch, 2019)

Figure 1 (above) shows an example Bluefetch enabled device running their software. Once the user signs in they are only able to access a small selection of applications relevant to them. Not only does this make the device easier to use, it also prevents unauthorised actions being performed as the UI is particularly locked down. All modifications are carried out remotely via support requests and cannot be performed by local staff members.

Costing

Additional costing would of course be incurred through the implementation of tablets into CC. These costs will come from elements such as the following:

- Tablet purchase agreements
- Software subscriptions
- Potential 4G subscriptions

Tablet purchase agreements should include tablets such as the following as they would be bet suited to this type of application. Tablets would need to be rugged and potentially water resistant as they would likely be used within close proximity to liquids. Tablet recommendations are as follows:

- Samsung Galaxy Tab 2 Active – 4G Enabled
- Panasonic Toughbook FZ-L1

Both tablets come with an IP water resistance rating making them able to withstand liquids. Both also include toughened cases as well as glass. As both tablets are also running Android, enterprise mobile device management software can be installed.

Tablet purchase costs vary depending on the device, however CC should expect costs of £300 per device unless a monthly subscription to a device supplier is setup. Between all 60 branches there will be 5 devices per branch meaning the total number of devices is 300.

Therefore, setup costs could potentially be:

$$300*300 = £90'000$$

Bluefetch subscription is then $36 per year per device (considering conversion rates) adding to a total annual cost of:

$$300*37*0.78 = £8'658$$

Costs for the database may also increase slightly due to extra tables and attributes being added therefore causing more data to then be held in the AWS database though the costs of extra storage will be far less than the yearly subscription costs.

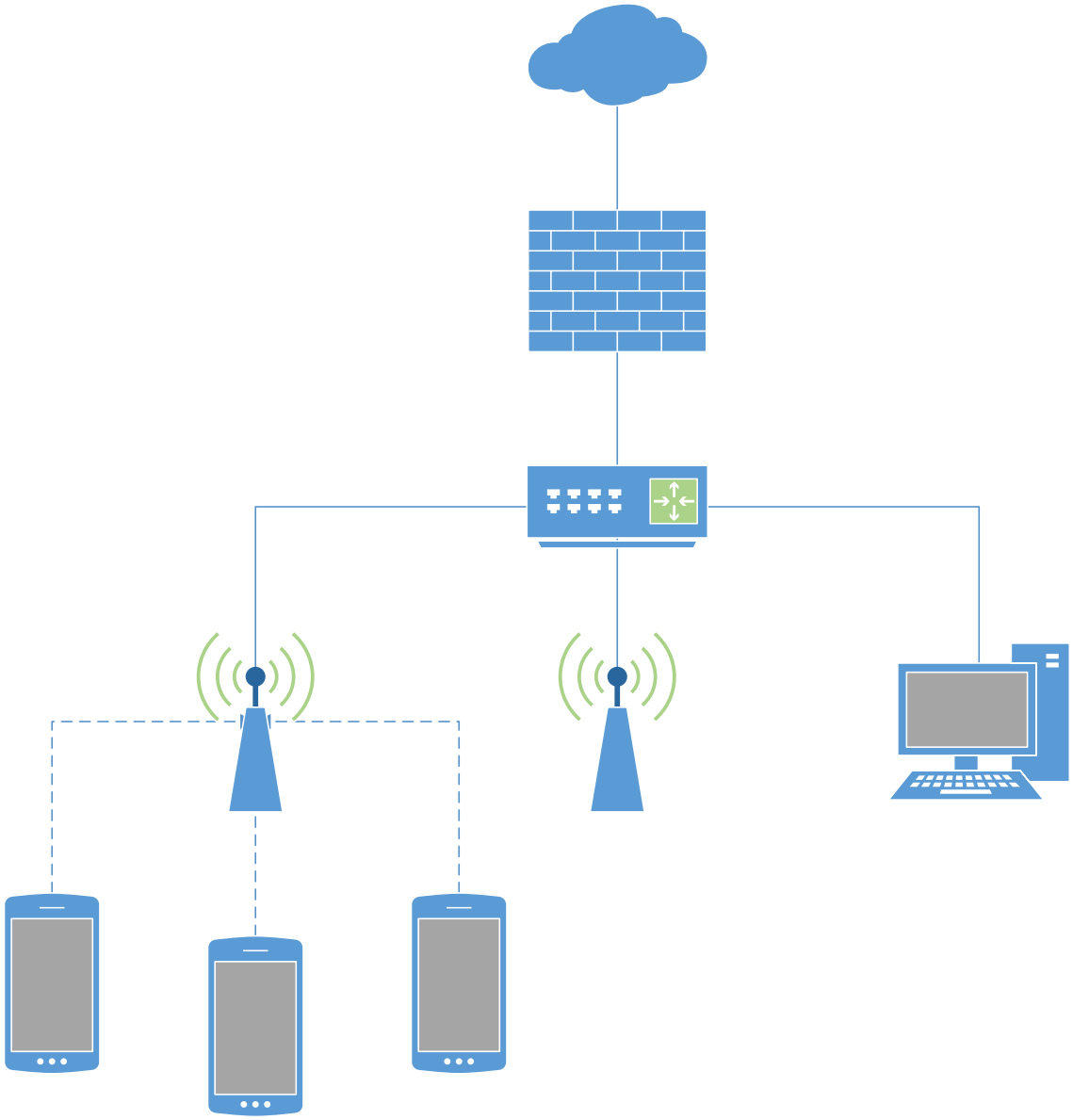
Database Connection

As the database is being hosted remotely on an AWS server(s) a stable connection would be required at all times. Stated in Q3 was the use of a private WAN connection along with an ADSL backup. These 2 combined connections methods along with a private and encrypted Wi-Fi network would allow for tablets to have constant access to the database. Through use of sophisticated access points, private networks can be setup. These networks can then be secured using WPA2 algorithms with long and complex passwords. The passwords could then be help by managerial staff to allow them to connect new and existing devices to the network. In order to prevent external access attempts the network can be hidden. This would then prevent unauthorised access attempts being made.

Another method of connection (as a failsafe) could be to make use of each tablets 4G interface. Data-only sim cards could be inserted and only used as and when a Wi-Fi connection is not possible. 4G connection speeds are improving drastically and so the speed deficits will not be as noticeable allowing orders to be pushed to the database within good time. This therefore would also mean that kitchen staff would also require a tablet to be able to see orders coming through, 4G would also need to be made available as they would still require live updates to view new orders.

Through using a web portal to use the databases features and make data requests means that data will not be stored on the tablets, not only does this keep the devices clutter free, as previously mentioned this also makes the devices less of a threat if they are stolen.

Network Diagram



The above network shows a mock layout of the CC branch network. A firewall and a router will be used to handle incoming and outgoing connections to the AWS MySQL database. Within the network multiple access points will be used to connect tablets to CC's DB. Desktop point of sales (POS) will also be required on the bar and will be connected through ethernet as a failover method if the Wi-Fi were to drop out at any point.

User Experience

Users would not see much of a difference in the way that they use the ordering system, this is because the web portal would be identical to the one used on the standard POS systems. The main differences would be that orders would need to be exclusively cashless and rely on customers having debit/credit cards in order to pay for goods. If a customer has cash the assistant should then instruct the customer to order at the bar as standard POS systems would have a cash drawer. The GUI should be identical (other than the feature previously stated), this would then mean minimal re-training would be required as staff members are then already familiar with the ordering system, not only would this save on training costs, it would also mean that more orders can be processed as staff members already know how to use the ordering system despite it being on a mobile POS device.

Q6 Queries

Sample Data and Table Design

Data implemented was generated using Mockaroo, this tool allows developers to define all entity attributes and generate data to be imputed within a live database for test purposes. Data imputed can later on be deleted when the database will become live and operational. Example data for each of the CC database entities is displayed below. Data types are also defined through use of a 'Describe' statement through MariaDB's command line interface. Data was then checked using Atom's addon for SQL encoding.

Table designs show in the following sections show the modified EERD layout discussed in Q4 optimization. The main aim of the design modification was to improve form and function as well as allowing queries to become simpler.

Branch

Table Design:

```
+----------+-------------+------+-----+---------+----------------+
| Field    | Type        | Null | Key | Default | Extra          |
+----------+-------------+------+-----+---------+----------------+
| branchID | int(11)     | NO   | PRI | NULL    | auto_increment |
| name     | varchar(45) | NO   |     | NULL    |                |
| address  | varchar(45) | NO   |     | NULL    |                |
| postCode | varchar(8)  | NO   | UNI | NULL    |                |
+----------+-------------+------+-----+---------+----------------+
```

Sample Data (10 Rows Shown):

```
insert into Branch (name, address, postCode) values ('Hošťka', 'Sunfield', 1);
insert into Branch (name, address, postCode) values ('Baiyushan', 'Linden', 2);
insert into Branch (name, address, postCode) values ('Pueblo Nuevo', 'Magdeline', 3);
insert into Branch (name, address, postCode) values ('Fresnes', 'Birchwood', 4);
insert into Branch (name, address, postCode) values ('Jiamaying', '5th', 5);
insert into Branch (name, address, postCode) values ('Sinop', 'Magdeline', 6);
insert into Branch (name, address, postCode) values ('Hezhang', 'Cardinal', 7);
insert into Branch (name, address, postCode) values ('Zelenchukskaya', 'Talmadge', 8);
insert into Branch (name, address, postCode) values ('Kuzovatovo', 'Northport', 9);
insert into Branch (name, address, postCode) values ('Ban Na', 'Menomonie', 10);
```

Post codes are defined as integers as opposed to genuine codes as Mockaroo doesn't support this function. Therefore, row numbers have been used to ensure that each branch has a unique entry for this attribute.

Shift

Design:

Field	Type	Null	Key	Default	Extra
shiftID	int(11)	NO	PRI	NULL	auto_increment
name	varchar(10)	NO		NULL	
totalHour	int(11)	NO		NULL	
breakTime	decimal(10,0)	YES		NULL	

Sample Data:

```
insert into Shift (name, totalHour, breakTime) values ('Morning', 6, 1.25);
insert into Shift (name, totalHour, breakTime) values ('Early', 4, 1.33);
insert into Shift (name, totalHour, breakTime) values ('Close', 6, 0.33);
insert into Shift (name, totalHour, breakTime) values ('Early', 5, 0.67);
insert into Shift (name, totalHour, breakTime) values ('Early', 8, 0.21);
insert into Shift (name, totalHour, breakTime) values ('Lunch', 6, 0.52);
```

Product

Design:

Field	Type	Null	Key	Default	Extra
prodID	int(11)	NO	PRI	NULL	auto_increment
Type	varchar(5)	NO		NULL	
Name	varchar(45)	YES	MUL	NULL	
Calorie	varchar(45)	YES		NULL	
Description	varchar(45)	YES		NULL	
Price	varchar(45)	YES		NULL	
Alcohol	varchar(45)	YES	MUL	NULL	

Sample Data (10 rows shown):

```
insert into Product (Type, Name, Calorie, Description, Price, Alcohol) values ('Food', 'Wine - Ruffino Chianti Classico', 540, 'Proin interdum mauris non ligula pellentesque ultrices.', 9.08, false);
insert into Product (Type, Name, Calorie, Description, Price, Alcohol) values ('Main', 'Rice - 7 Grain Blend', 533, 'Aenean lectus.', 6.77, false);
insert into Product (Type, Name, Calorie, Description, Price, Alcohol) values ('Food', 'Wine - George Duboeuf Rose', 72, 'Quisque ut erat.', 5.45, false);
insert into Product (Type, Name, Calorie, Description, Price, Alcohol) values ('Food', 'Berry Brulee', 128, 'Integer ac neque.', 9.39, false);
insert into Product (Type, Name, Calorie, Description, Price, Alcohol) values ('Alcoholic Drink', 'Juice - Apple, 1.36l', 260, 'Praesent blandit lacinia erat.', 1.36, true);
insert into Product (Type, Name, Calorie, Description, Price, Alcohol) values ('Soft Drink', 'Sauce - Ranch Dressing', 227, 'Proin eu mi.', 3.09, true);
insert into Product (Type, Name, Calorie, Description, Price, Alcohol) values ('Alcoholic Drink', 'Turnip - Wax', 654, 'Nulla justo.', 6.21, true);
insert into Product (Type, Name, Calorie, Description, Price, Alcohol) values ('Snack', 'Pastry - Plain Baked Croissant', 207, 'Sed accumsan felis.', 4.98, false);
insert into Product (Type, Name, Calorie, Description, Price, Alcohol) values ('Snack', 'Versatainer Nc - 9388', 27, 'Mauris enim leo, rhoncus sed, vestibulum sit amet, cursus id, turpis.', 6.39, true);
insert into Product (Type, Name, Calorie, Description, Price, Alcohol) values ('Soft Drink', 'Placemat - Scallop, White', 12, 'In est risus, auctor sed, tristique in, tempus sit amet, sem.', 6.54, false)
```

Category

Design:

Field	Type	Null	Key	Default	Extra
catID	int(11)	NO	PRI	NULL	auto_increment
Description	varchar(45)	YES		NULL	
Product_prodID	int(11)	NO	MUL	NULL	

Sample Data (10 rows shown):

```
insert into Category (Description, Product_ProdID) values ('Lorem ipsum dolor sit amet, consectetuer adipiscing elit.', 694);
insert into Category (Description, Product_ProdID) values ('Aenean lectus.', 720);
insert into Category (Description, Product_ProdID) values ('In hac habitasse platea dictumst.', 543);
insert into Category (Description, Product_ProdID) values ('Aliquam sit amet diam in magna bibendum imperdiet.', 80);
insert into Category (Description, Product_ProdID) values ('Nulla mollis molestie lorem.', 70);
insert into Category (Description, Product_ProdID) values ('Vivamus metus arcu, adipiscing molestie, hendrerit at, vulputate vitae, nisl.', 210);
insert into Category (Description, Product_ProdID) values ('Vestibulum rutrum rutrum neque.', 148);
insert into Category (Description, Product_ProdID) values ('In eleifend quam a odio.', 993);
insert into Category (Description, Product_ProdID) values ('Phasellus id sapien in sapien iaculis congue.', 274);
insert into Category (Description, Product_ProdID) values ('Nullam varius.', 588);
```

Staff

Design:

Field	Type	Null	Key	Default	Extra
staffID	int(11)	NO	PRI	NULL	auto_increment
home_branchID	int(11)	YES		NULL	
fName	varchar(45)	YES		NULL	
lName	varchar(45)	YES	MUL	NULL	
address	varchar(45)	YES		NULL	
postCode	varchar(8)	YES		NULL	
natInsNo	varchar(10)	YES	UNI	NULL	
position	varchar(30)	YES		NULL	
contType	varchar(20)	YES		NULL	
hourRate	decimal(10,2)	YES		NULL	

Sample Data (10 rows shown):

```
insert into Staff (home_branchID, fName, lName, address, postCode, natInsNo, position, contType, hourRate) values (5, 'Sisely', 'Hould', 'American Ash', 1, 1, 'Floor', 'Part', 8.75);
insert into Staff (home_branchID, fName, lName, address, postCode, natInsNo, position, contType, hourRate) values (34, 'Kacy', 'Gerlts', 'Harper', 2, 2, 'Chef', 'Part', 7.81);
insert into Staff (home_branchID, fName, lName, address, postCode, natInsNo, position, contType, hourRate) values (16, 'Dagny', 'Measham', 'Washington', 3, 3, 'Floor', 'Part', 7.53);
insert into Staff (home_branchID, fName, lName, address, postCode, natInsNo, position, contType, hourRate) values (44, 'Vevay', 'Rispine', 'Mallory', 4, 4, 'Manager', 'Full', 9.87);
insert into Staff (home_branchID, fName, lName, address, postCode, natInsNo, position, contType, hourRate) values (41, 'Ruthi', 'Dunnion', 'La Follette', 5, 5, 'Bar', 'Full', 10.52);
insert into Staff (home_branchID, fName, lName, address, postCode, natInsNo, position, contType, hourRate) values (2, 'Banky', 'McFadin', 'Jenifer', 6, 6, 'Bar', 'Full', 9.29);
insert into Staff (home_branchID, fName, lName, address, postCode, natInsNo, position, contType, hourRate) values (7, 'Arlin', 'Aspy', 'Raven', 7, 7, 'Chef', 'Part', 11.37);
insert into Staff (home_branchID, fName, lName, address, postCode, natInsNo, position, contType, hourRate) values (10, 'Melinda', 'Towsey', 'Delladonna', 8, 8, 'Bar', 'Full', 6.83);
insert into Staff (home_branchID, fName, lName, address, postCode, natInsNo, position, contType, hourRate) values (3, 'Brittan', 'Abbotts', 'Sundown', 9, 9, 'Bar', 'Part', 10.75);
insert into Staff (home_branchID, fName, lName, address, postCode, natInsNo, position, contType, hourRate) values (2, 'Saleem', 'Elsmore', 'Vidon', 10, 10, 'Bar', 'Part', 8.59);
```

Order Line

Design:

Field	Type	Null	Key	Default	Extra
orderLineID	int(11)	NO	PRI	NULL	auto_increment
Product_prodID	int(11)	NO	MUL	NULL	
quant	int(11)	YES		NULL	
linePrice	decimal(10,2)	YES		NULL	
tableOrder_ordID	varchar(45)	NO	MUL	NULL	

Sample Data (10 rows shown):

```
insert into orderLine (Product_prodID, quant, linePrice, tableOrder_ordID) values (568, 1, 8.88, 251);
insert into orderLine (Product_prodID, quant, linePrice, tableOrder_ordID) values (258, 3, 8.6, 928);
insert into orderLine (Product_prodID, quant, linePrice, tableOrder_ordID) values (781, 1, 4.06, 544);
insert into orderLine (Product_prodID, quant, linePrice, tableOrder_ordID) values (662, 2, 5.64, 166);
insert into orderLine (Product_prodID, quant, linePrice, tableOrder_ordID) values (432, 3, 4.0, 622);
insert into orderLine (Product_prodID, quant, linePrice, tableOrder_ordID) values (362, 1, 2.17, 714);
insert into orderLine (Product_prodID, quant, linePrice, tableOrder_ordID) values (545, 3, 8.28, 465);
insert into orderLine (Product_prodID, quant, linePrice, tableOrder_ordID) values (192, 2, 4.2, 278);
insert into orderLine (Product_prodID, quant, linePrice, tableOrder_ordID) values (382, 1, 5.42, 150);
insert into orderLine (Product_prodID, quant, linePrice, tableOrder_ordID) values (655, 1, 2.0, 438);
```

Staff Shift:

Design:

```
+-----------------+------------+------+-----+---------+----------------+
| Field           | Type       | Null | Key | Default | Extra          |
+-----------------+------------+------+-----+---------+----------------+
| staffShiftID    | int(11)    | NO   | PRI | NULL    | auto_increment |
| overtime        | tinyint(1) | NO   |     | NULL    |                |
| Staff_staffID   | int(11)    | NO   | MUL | NULL    |                |
| Shift_shiftID   | int(11)    | NO   | MUL | NULL    |                |
| Branch_branchID | int(11)    | NO   | MUL | NULL    |                |
+-----------------+------------+------+-----+---------+----------------+
```

Sample Data (10 rows shown):

```
insert into staffShift (overtime, Staff_staffID, Shift_shiftID, Branch_branchID) values (true, 417, 1, 39);
insert into staffShift (overtime, Staff_staffID, Shift_shiftID, Branch_branchID) values (false, 373, 2, 5);
insert into staffShift (overtime, Staff_staffID, Shift_shiftID, Branch_branchID) values (true, 705, 5, 17);
insert into staffShift (overtime, Staff_staffID, Shift_shiftID, Branch_branchID) values (true, 292, 3, 22);
insert into staffShift (overtime, Staff_staffID, Shift_shiftID, Branch_branchID) values (false, 485, 6, 44);
insert into staffShift (overtime, Staff_staffID, Shift_shiftID, Branch_branchID) values (false, 393, 6, 25);
insert into staffShift (overtime, Staff_staffID, Shift_shiftID, Branch_branchID) values (true, 415, 2, 17);
insert into staffShift (overtime, Staff_staffID, Shift_shiftID, Branch_branchID) values (true, 845, 6, 30);
insert into staffShift (overtime, Staff_staffID, Shift_shiftID, Branch_branchID) values (false, 193, 2, 18);
insert into staffShift (overtime, Staff_staffID, Shift_shiftID, Branch_branchID) values (false, 848, 3, 45);
```

Table Order:

Design:

```
+-----------------+-------------+------+-----+---------+-------+
| Field           | Type        | Null | Key | Default | Extra |
+-----------------+-------------+------+-----+---------+-------+
| ordID           | varchar(45) | NO   | PRI | NULL    |       |
| ordTime         | datetime    | YES  |     | NULL    |       |
| tableNo_tableID | int(11)     | NO   | MUL | NULL    |       |
+-----------------+-------------+------+-----+---------+-------+
```

Sample Data (10 rows shown):

```
insert into tableOrder (ordID, ordTime, tableNo_tableID) values (1, '2014-01-01 00:00:00', 6);
insert into tableOrder (ordID, ordTime, tableNo_tableID) values (2, '2014-01-02 00:00:00', 14);
insert into tableOrder (ordID, ordTime, tableNo_tableID) values (3, '2014-01-03 00:00:00', 16);
insert into tableOrder (ordID, ordTime, tableNo_tableID) values (4, '2014-01-04 00:00:00', 11);
insert into tableOrder (ordID, ordTime, tableNo_tableID) values (5, '2014-01-05 00:00:00', 18);
insert into tableOrder (ordID, ordTime, tableNo_tableID) values (6, '2014-01-06 00:00:00', 18);
insert into tableOrder (ordID, ordTime, tableNo_tableID) values (7, '2014-01-07 00:00:00', 19);
insert into tableOrder (ordID, ordTime, tableNo_tableID) values (8, '2014-01-08 00:00:00', 22);
insert into tableOrder (ordID, ordTime, tableNo_tableID) values (9, '2014-01-09 00:00:00', 13);
insert into tableOrder (ordID, ordTime, tableNo_tableID) values (10, '2014-01-10 00:00:00', 28);
```

Table N.o:

Design:

```
+-----------------+---------+------+-----+---------+----------------+
| Field           | Type    | Null | Key | Default | Extra          |
+-----------------+---------+------+-----+---------+----------------+
| tableID         | int(11) | NO   | PRI | NULL    | auto_increment |
| Branch_branchID | int(11) | NO   | MUL | NULL    |                |
| intTableID      | int(11) | NO   | UNI | NULL    |                |
+-----------------+---------+------+-----+---------+----------------+
```

Sample Data (10 rows shown):

```
insert into tableNo (Branch_branchID, intTableID) values (40, 1);
insert into tableNo (Branch_branchID, intTableID) values (31, 2);
insert into tableNo (Branch_branchID, intTableID) values (18, 3);
insert into tableNo (Branch_branchID, intTableID) values (13, 4);
insert into tableNo (Branch_branchID, intTableID) values (47, 5);
insert into tableNo (Branch_branchID, intTableID) values (52, 6);
insert into tableNo (Branch_branchID, intTableID) values (29, 7);
insert into tableNo (Branch_branchID, intTableID) values (1, 8);
insert into tableNo (Branch_branchID, intTableID) values (4, 9);
insert into tableNo (Branch_branchID, intTableID) values (39, 10);
insert into tableNo (Branch_branchID, intTableID) values (41, 11);
```

Queries

Queries were written in Atom as it supports MySQL syntax allowing for code to be written simpler and in the correct format. Putty was then used to push data to a virtual machine which hosts the database and therefore is where the query data comes from.

Query 1:

The aim of this query is to determine which products are most commonly purchased together, this can be seen as a form of market intelligence as the following conclusions can be bought together:

- Where to form multibuy offers
- What products to generate 'meal deals' on
- Where product prices may need to be increased to generate more income

A limit of 15 has been imposed due to the number of records held in the dataset, this limit can be increased to aid with determination of popular product combinations.

Marketing managers companywide would find this information of great use, not only for the reasons described above but to also introduced new products based on those that are frequently purchased.

The query code is as follows:

```
-->selection of top 10 combinations of products
SELECT PR1.Name as "Product 1", PR2.Name as "Product 2", COUNT(*) AS "Combo Quantity"
FROM orderLine AS OL1
INNER JOIN orderLine AS OL2
ON OL1.tableOrder_ordID = OL2.tableOrder_ordID
AND OL1.Product_prodID > OL2.Product_prodID
INNER JOIN Product AS PR1
ON OL1.Product_prodID = PR1.prodID
INNER JOIN Product AS PR2
ON OL2.Product_prodID = PR2.prodID
GROUP BY PR1.Name, PR2.Name
HAVING COUNT(*)>=1
ORDER BY COUNT(*)DESC
LIMIT 15;
```

When used in Putty the results returned were as follows:

Product 1	Product 2	Combo Quantity
Vinegar - White	Pail For Lid 1537	2
Vinegar - White	Strawberries - California	2
Vinegar - White	Juice - Apple, 1.36l	2
Veal - Striploin	Sea Bass - Whole	1
Bread Cranberry Foccacia	Mushroom - Chanterelle Frozen	1
Red Currants	Parsley - Dried	1
Spinach - Spinach Leaf	Pike - Frozen Fillet	1
Rabbit - Saddles	Plastic Arrow Stir Stick	1
Nut - Walnut, Chopped	Kohlrabi	1
Mix - Cocktail Ice Cream	Cheese - Brie	1
Salmon - Atlantic, Fresh, Whole	Wine - Valpolicella Masi	1
Dip - Tapenade	Liquid Aminios Acid - Braggs	1
Cinnamon - Stick	Chicken Giblets	1
Pepper - White, Ground	Thyme - Lemon, Fresh	1
Energy Drink - Franks Pineapple	Wine - Jackson Triggs Okonagan	1

Relational Algebra & Query Tree:

$$\pi\ PR1.Name, PR2.Name, COUNT(*)((orderLine \bowtie orderLine.tableOrder\_ordID = orderLine.tableOrder\_ordID \wedge orderLine.Product\_prodID \geq orderLine.Product\_prodID \bowtie Product\ orderLine.Product\_prodID = Product.prodID \bowtie Product\ orderLine.Product\_prodID = Product.prodID (\sigma\ COUNT(*) \geq 1)))$$

Query 2:

This query is designed to determine the busiest date for all branches in terms of the higher amount of money made per branch. Use of a temporary table was necessary to further manipulate the results to ensure usefulness of the query. The query helps the business to then understand which branches are busiest on what particular date. In future this data could be used to help plan shifts and determine the peak revenue generated from each branch.

End users for this query would almost exclusively be for shift manager and those at higher levels of management. This would also aid with HR management and allocation if more staff members are required at different branches.

The code is as follows:

```
DROP  TABLE IF EXISTS BRANCHDAYSALES;
CREATE TABLE BRANCHDAYSALES
SELECT BR.branchID as "BID", BR.name as "BRName", CAST(ordTime AS DATE) as "Date", SUM(linePrice) as "Total"
FROM tableOrder AS O
INNER JOIN orderLine AS OL
ON O.ordID = OL.tableOrder_ordID
INNER JOIN tableNo AS TN
ON O.tableNo_tableID = TN.tableID
INNER JOIN Branch AS BR
ON TN.Branch_branchID = BR.branchID
GROUP BY BR.Name, ordTime;

SELECT BHS.BID as "Branch ID", BHS.BRName as "Branch Name", BHS.Date as "Busiest Date", BHS.TOTAL as "Total Sales" FROM BRANCHDAYSALES AS BHS
INNER JOIN(SELECT BRName,MAX(TOTAL) AS TOTAL FROM BRANCHDAYSALES
GROUP BY BRName)AS MAX_SALES ON BHS.BRName = MAX_SALES.BRName AND BHS.TOTAL = MAX_SALES.TOTAL
ORDER BY MAX_SALES.TOTAL DESC;

DROP TABLE BRANCHDAYSALES;
```

When executed the following data is returned:

```
+-----------+-------------------+--------------+-------------+
| Branch ID | Branch Name       | Busiest Date | Total Sales |
+-----------+-------------------+--------------+-------------+
|        13 | Xiangshui         | 2014-09-23   |       31.88 |
|        58 | Colca             | 2014-01-14   |       28.93 |
|        50 | Épinal            | 2015-11-24   |       28.22 |
|        41 | Canela            | 2014-10-15   |       26.26 |
|        60 | Saurama           | 2015-06-01   |       25.84 |
|         4 | Fresnes           | 2015-04-24   |       25.35 |
|         1 | Hošťka            | 2015-10-03   |       23.13 |
|        33 | Vsetín            | 2015-12-06   |       23.04 |
|        47 | Baiyun            | 2014-11-18   |       22.61 |
|        31 | Tamontaka         | 2015-10-10   |       22.35 |
|        39 | Uk                | 2016-07-19   |       21.63 |
|        15 | Puchong           | 2016-04-08   |       21.22 |
|        55 | Thị Trấn Lâm      | 2015-12-26   |       20.89 |
|        46 | Gaosheng          | 2014-05-18   |       20.30 |
|        18 | Lanshan           | 2015-11-17   |       19.88 |
|        52 | Berlin            | 2014-01-11   |       19.83 |
|        49 | Íquira            | 2014-01-30   |       19.47 |
|        22 | Sa'sa'            | 2014-05-01   |       18.97 |
|        40 | Gulu              | 2015-08-09   |       18.46 |
|        26 | Oceanside         | 2014-09-14   |       18.05 |
|        51 | Darvi             | 2015-04-10   |       17.23 |
|        57 | Wanga             | 2015-02-25   |       16.04 |
|        29 | Dashiqiao         | 2015-02-28   |       15.94 |
|         6 | Sinop             | 2016-08-29   |       14.57 |
|         8 | Zelenchukskaya    | 2014-02-08   |       14.29 |
+-----------+-------------------+--------------+-------------+
25 rows in set (0.00 sec)

MariaDB [ADCON]>
MariaDB [ADCON]> DROP TABLE BRANCHDAYSALES;
```

Relational Algebra & Query Tree:

Part 1 of code: π BR.branchID, BR.name, ordTime, SUM(linePrice)((orderLine × order.ordID = OL.tableOrder_ordID × O.tableNo_tableID = tableNo.tableID × TN.Branch_branchID = BR.branchID))

Part 2 of code: π BHS.BID,BHS.BRName,BHS.Date,BHS.TOTAL × ((σ BRANCHDAYSALES π BRName σ MAX(TOTAL)))

Query 3:

In order to determine which products to put on special offer at different times of the year it is important to first determine the most popular products within a certain timeframe. This query lists the top 10 most popular 'Main' type foods between January 1st 2014 and 2016. Along with the product type, dates can also be changed to get a more accurate and precise representation.

This would again be useful for marketing managers as they would then know which campaigns have been successful and what products they should focus on in order to boost sales on particular products.

The code is as follows:

```sql
-->selection of top 10 most popular products within a set timeframe
SELECT CONCAT(P.Name ," // "  , P.Description) as "Product Name & Description", P.Price as "Price", SUM(OL.quant) as "Quantity Sold"
FROM Product AS P
JOIN orderLine as OL
ON P.prodID = OL.Product_prodID
JOIN tableOrder as TA
ON TA.ordID = OL.tableOrder_ordID
WHERE TA.ordTime BETWEEN CAST('2014-01-01' AS DATE) AND CAST('2016-01-01' AS DATE) AND P.Type = 'Main'
GROUP BY P.prodID
ORDER BY OL.quant DESC
LIMIT 10;
```

Results returned:

Product Name & Description	Price	Quantity Sold
Dip - Tapenade // Curabitur convallis.	4.77	6
Shrimp - Black Tiger 13/15 // Aliquam erat volutpat.	2.08	3
Sour Puss Sour Apple // Nulla nisl.	5.75	6
Rice - 7 Grain Blend // Aenean lectus.	6.77	3
Cheese - Mozzarella, Shredded // Nulla tempus.	7.73	8
Cabbage - Nappa // Morbi odio odio, elementum eu, interdum eu, t	4.33	3
Butter Ripple - Phillips // Cras pellentesque volutpat dui.	7.35	3
Flour - So Mix Cake White // Mauris enim leo, rhoncus sed, vestibulum sit	3.26	3
Fondant - Icing // Vivamus metus arcu, adipiscing molestie, hend	8.92	3
Chevere Logs // Vestibulum ante ipsum primis in faucibus orci	1.93	5

Relational Algebra & Query Tree:

$$\pi\ p.name, p.description, P.PRICE, SUM(OL.quant)((ORDERLINE \bowtie P.prodID = OL.Product\_productID \bowtie TABLEORDER.ordID = OL.tableOrder\_ordID(\sigma\ TA.ordTime \geq '2014\text{-}01\text{-}01' \wedge TA.ordTime \leq 2016\text{-}01\text{-}01 \wedge (\sigma\ P.type = 'Main')))$$

Query 4:

This query is designed to determine the total sales for each staff member. Where clauses are used to define the date of a staff members shift and the branch in question. This would be useful to branch directors/managers as they are then able to determine the highest performing staff member (on the basis of sales) on a particular day at a particular branch. This may also aid in determining the size of teams to be on shift simultaneously.

A view has been implemented as the information presented is viewed directly by staff members within management. Views can be configured with built in access control thus preventing access to the data by any other staff member other than those defined 'management'. Further protection can also be provided by only allowing branch managers to view their own data and not that of any other branch. This would then reduce competitively inter-branch.

The code used is as follows:

```
CREATE VIEW STAFF_SALES as(
SELECT CONCAT(S.fName, " ", S.lName) as "Name", SH.Branch_branchID as "Branch Worked", SUM(OL.linePrice) as "Total Revenue"
FROM Staff as S
JOIN staffShift as SH
ON S.staffID = SH.Staff_staffID
JOIN Branch as B
ON SH.Branch_branchID = B.branchID
JOIN tableNo as TN
ON B.branchID = TN.Branch_branchID
JOIN tableOrder as TABO
ON TN.tableID = TABO.tableNo_tableID
JOIN orderLine as OL
ON TABO.ordID = OL.tableOrder_ordID
WHERE SH.shiftDate = CAST('2019/01/10'AS DATE) AND B.branchID = '18'
GROUP BY S.staffID
ORDER BY 'Total Revenue' DESC);
```

Results returned from this query are as follows:

```
MariaDB [ADCON]> describe STAFF_SALES;
+---------------+---------------+------+-----+---------+-------+
| Field         | Type          | Null | Key | Default | Extra |
+---------------+---------------+------+-----+---------+-------+
| Name          | varchar(91)   | YES  |     | NULL    |       |
| Branch Worked | int(11)       | NO   |     | NULL    |       |
| Total Revenue | decimal(32,2) | YES  |     | NULL    |       |
+---------------+---------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

Relational Algebra & Query Tree:

π S.fName, S.lName, SH.Branch_branchID, SUM(OL.linePrice)((staffShift ⋈ Staff.staffID = SH.Staff_staffID ⋈ SH.Branch_branchID = Branch.branchID ⋈ B.tableNo = tableNo.Branch_branchID ⋈ TN.tableID = tableOrder.tableNo_tableID ⋈ TABO.ordID = orderLine.tableOrder.ordID)(σ SH.shiftDate = '26-01-10 ∧ B.branchID = '18))

Query 5:

This query is used to determine daily stock recalculation records. As previously described within the assumptions for CC's DB stock systems are out of the scope of this new DB implementation and so only sales figures are provided. A separate stock system (whether this be paper or digital in branch) would be used to determine when products need to be ordered. The query can be altered to change the date in question or the branch in question to make personalized reports for each branch.

This query would be useful for stock controllers as well as branch managers. Stock controllers would use the information to complete new orders as well as to let managers know which products are more likely to sell based on the frequency of re-ordering.

The code is as follows:

```
75  -->Count of all product sales totals for each day at a particualr branch used for restocking calculations
76  SELECT CONCAT(P.Name ," // "  , P.Description) as "Product Name & Description", SUM(OL.quant) as "Quantity Sold"
77  FROM Product AS P
78  JOIN orderLine as OL
79  ON P.prodID = OL.Product_prodID
80  JOIN tableOrder as TA
81  ON TA.ordID = OL.tableOrder_ordID
82  INNER JOIN tableNo as TABNO
83  ON TABNO.tableID = TA.tableNo_tableID
84  INNER JOIN Branch AS B
85  ON TABNO.Branch_branchID = B.branchID
86  WHERE TA.ordTime = CAST('2015-06-12' AS DATE) AND  B.branchID = '1'
87  GROUP BY P.prodID
88  ORDER BY OL.quant DESC;
```

The results returned are as follows:

```
+-----------------------------------------------------------+---------------+---------------------+
| Product Name & Description                                | Quantity Sold | ordTime             |
+-----------------------------------------------------------+---------------+---------------------+
| Pie Pecan // Cras in purus eu magna vulputate luctus.     |             3 | 2015-06-12 00:00:00 |
| Breadfruit // In est risus, auctor sed, tristique in, tempu |             1 | 2015-06-12 00:00:00 |
+-----------------------------------------------------------+---------------+---------------------+
2 rows in set (0.00 sec)
```

Relational Algebra:

π P.Name, P.Description, SUM(OL.quant)((orderLine ⋈ Product.prodID = orderLine.Product_prodID ⋈ tableOrder.ordID = OL.tableOrder_ordID × tableNo.tableID = TA.tableNo_tableID × Branch.branchID = TABNO.tableID σ(TA.ordTime = '2015-06-12' ∧ B.branchID ='1'(BRANCH))))

Query 6:

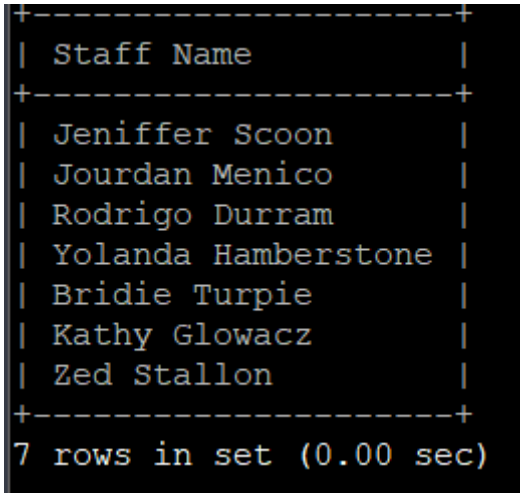
Query 6 is used to determine staff members who have not sold a particular product. This would be useful when head office targets branches to sell products designated within 'Deal Of the Month' offers. Those staff members who appear on the query are those who have not sold the product and so can be given 'that extra push' by branch managers to upsell the product(s) in question.

Bonuses could also be derived from this query, those included in the list would not be entitled to a bonus whereas those not included would be entitled. Although this decision would be down to the branch managers and HR staff.

The code is as follows:

```
101  SELECT CONCAT(S.fName, " ", S.lName) as "Staff Name"
102  FROM Staff as S
103  JOIN staffShift as SH
104  ON S.staffID = SH.Staff_staffID
105  JOIN Branch as B
106  ON SH.Branch_branchID = B.branchID
107  JOIN tableNo as TN
108  ON B.branchID = TN.Branch_branchID
109  JOIN tableOrder as TABO
110  ON TN.tableID = TABO.tableNo_tableID
111  JOIN orderLine as OL
112  ON TABO.ordID = OL.tableOrder_ordID
113  LEFT JOIN Product as PR
114  ON OL.Product_prodID = PR.prodID
115   AND PR.Name = "Impossible Burger"
116   WHERE S.home_branchID =1
117   AND PR.ProdID IS NULL
118  GROUP BY S.staffID;
```

The results returned (discovering names of staff members based at branch '1' who haven't sold product 'Impossible Burger').



```
+--------------------+
| Staff Name         |
+--------------------+
| Jeniffer Scoon     |
| Jourdan Menico     |
| Rodrigo Durram     |
| Yolanda Hamberstone |
| Bridie Turpie      |
| Kathy Glowacz      |
| Zed Stallon        |
+--------------------+
7 rows in set (0.00 sec)
```

Relational Algebra:

π S.fName, S.lName ((staffShift.Staff_staffID = Staff.staffID ⋈ SH.Branch_branchID = Branch.branchID ⋈ B.branchID = tableNo.Branch_branchID ⋈ TN.tableID = tableOrder.tableNo_tableID ⋈ TABO.ordID = orderLine.tableOrder_ordID ⋉ OL.Product_prodID = Product.prodID ∧ PR.name = 'Impossible Burger' σ(S.home_branchID =1 ∧ PR.ProdID = NULL)))

References

Bluefetch. (2019). Support Duo [Image]. Retrieved from https://bluefletch.com/wp-content/uploads/2018/02/support-duo-on-tc70-948x1024.png

DBoptimiser – mention the use of the tool within an enterprise environment – speculative

Ghemri, L. (2019). The Relational Algebra [Ebook] (p. 1). Texas Southern University. Retrieved from http://cs.tsu.edu/ghemri/CS681/ClassNotes/Relat_Alg1.pdf