Group 1:
Pool it Together

Huibo Jia
Cuicui Lu
Sylvester Setio
Free Burton

Mission Statement:

Our mission is to make the management of a seasonal business smoother for a family-owned pool store that both sells chemicals, pool toys, and other merchandise as well as provide service on pools to create a well-rounded business model. Our application will give a real-time view of inventory and contractor management so that data can be seen and acted upon by managers with ease. This will improve efficiency within the business, reduce costs for contractors, and improve relationships between the business and its customers by fulfilling projects in a time-efficient manner.

Problem Statement:

For a swimming pool company, business is seasonal and with that comes unique challenges with project and inventory management. Retaining employees is difficult with the seasonal nature of the business, so they usually rely upon contractors to perform service work for customers. As a small mom and pop shop, having effective reporting tools in these often complex business structures is far and few between. Keeping track of swimming pool details for each customer, their past services and product purchases, and which contractor worked on those projects requires a lot of organization. Without connecting data sources, it will be difficult for managers to view the overall scope and functions within the business.

This is a problem that can affect the efficiency of the organization, and beyond that it can also affect the ability to hire capable contractors for service jobs. In this kind of business, it is especially important to build trust among contractors with expedited payment so that our business has reliable options to perform all the jobs our customers need. Especially when we have multiple large projects going on at one time, it is difficult to keep track of which jobs are being fulfilled and which jobs are yet to be completed, and the payments associated with each. In addition, considering the nature of working on a swimming pool, some service jobs can be quite pricey resulting in longer waits for a customer to fully pay off the service fee. If we fail to organize the structure of the business effectively, we can lose contractors to larger businesses, showing the importance of a mutual trust that keeps contractors from working for our competitors and leaving us with no one to complete projects. Our database will keep track of past and future services to make sure all projects are paid off and contractors get their money in a timely fashion.

In the retail store, it is not unlikely for data to be entered incorrectly when someone is making a purchase or returning an item, or to run out of stock on items that are particularly popular. Other problems arise often, with the store being overstaffed or understaffed and high priority tasks are sometimes left unfinished when it is pertinent that they be handled immediately. Keeping enough parts in stock for customers that visit the store with unique questions about their swimming pool can be difficult as every year there are new models and different parts to work with each, so ordering directly from the supplier to complete a project is a common occurrence.

These separate issues that happen on a daily basis can lead to jumbled data reporting in the moment, resulting in issues for managers that use the data to make decisions for the business. This all makes analyzing data on the back-end extremely difficult to keep the business running smoothly and to make any improvements beyond everyday functions. Our database aims to relieve all of these issues and boost overall business efficiency so that competing with larger businesses will no longer be as difficult. The database will keep track of inventory, sales orders, service projects, and the billing rate for each contractor. Managers can then use our app to see an overview of how everything is running to make sure they are on track and to also find areas for improvement. This can all be done by looking at historical or real-time data, depending on the managers' needs.

Business Rules

Membership Privileges

In order for a customer to be considered a member of *Pool it Together*, they must meet one of the following requirements: they have purchased at least one item in the retail store or they have ordered a service. In addition, members will have visited the store and filled out their personal information, including their full name, address, phone number, and email address. As a member of *Pool it Together*, customers will benefit from free water testing any time they visit the store. This is to increase traffic within the store and to limit the amount of people that will try and benefit from such a system without purchasing any products from the business. In addition to this, *Pool it Together* aims to reward the top five customers with rewards for their loyalty. This could include a number of things from free chemicals to a free pool inspection.

Swimming Pools

Swimming pool information will be completed on a case by case basis as a customers' needs expand over time. For instance, pool dimensions are not entirely useful unless completing a liner replacement, in which case that information will need to be filled out. The same goes for pumps and filters when they are in need of a replacement. This will help the company to improve efficiency when working on the pools of recurring customers. If the customer is a top five member, they will most likely have all swimming pool measurements and specifications filled out so that if they require service it can be dealt with quickly.

Contractors

Contractors will provide all the swimming pool services. Orders will accumulate as more pool services are worked on and completed. It is the company's responsibility to ensure that pricing is adequately competitive with market prices and profitable to both parties. This will be done in terms of previously negotiated prices with the contractors. *Pool it Together* provides the customer base and acts as the middle-man between customers and contractors in order to fill a customer's needs. For every reference we give to a contractor they will receive their designated fixed price for the work as we negotiate for higher margins with the customer based on terms of quality and speed of completion. It is the contractor's responsibility to manage the organization of future appointments so that they are completed in a timely manner for the customer. It is also their responsibility to maintain work at a level of quality that meets company standards. The monetary fault for not meeting standards falls on the contractor. When investigating past and future services, they must never match one another. Therefore, it is impossible to display services that have not yet been completed by removing observations that have been completed. This will keep the tables organized, however data must be maintained periodically to ensure deadlines are met.

Products & Employee Sales

Products are interrelated with purchases and sales orders. A precise inventory level can be found by aggregating the sum of these transactions. Sales orders reduce the quantity of a unit in inventory and a purchase order would increase the quantity of units in inventory. The product comes from the supplier and eventually will end up in the hands of the customer, so there are no other sources of products that *Pool it Together* provides. The price of an item from a supplier must not exceed the price it is sold for in the store. Most products aim to exceed a price margin of the product price from the supplier of at least 110%. This will ensure that the business keeps a high profit margin on normal business days. However, *Pool it Together* will occasionally have sales on products which may reduce price margins on some items, but the goal of making a profit on any item stays the same. This is also important when analyzing employee performance. If an employee performs extraordinarily well it would be commonplace to reward them for their efforts such as a end of the year bonus. This should drive them to repeat the same outstanding performance year after year.

Queries and Stored Procedures

1. Having multiple contractors and service appointments without any scheduling mechanisms can make the operations of a company work blindly. Through this stored procedure, we can pull up the name of any contractor and see whether the person is scheduled or not. If we were to schedule an additional appointment time for the person, we can order it as to not conflict with other appointments.

   /* Creating Stored Procedure to find contractor schedule */

```
DELIMITER //
CREATE PROCEDURE find_contractor_schedule
(IN name CHAR(20))
BEGIN
        select fs.service_date "Service Appointment Time",
          fs.estimated_lasting_time "Estimated Service Time (Hours)",
    fs.service_description "Service Description",
    concat(c.first_name, " " , c.last_name) "Contractor Name",
    cu.phone "Customer Phone Number",
    concat(l.address, " ", l.city, " ", l.state, " ", l.zip_code) "Customer Address"
        from ((FutureSchedule fs inner join Contractor c on fs.contractor_id = c.contractor_id) inner join
    Customer cu on fs.customer_id = cu.customer_id) inner join
    Location l on cu.location_id = l.location_id
        where concat(c.first_name, " ", c.last_name) = name
        Order by fs.service_date;
END //
DELIMITER ;

CALL find_contractor_schedule('Chelsea Jia');
```

   Output:

Service Appointment Time	Estimated Service Time (Hours)	Service Description	Contractor Name	Customer Phone Number	Customer Address
2020-02-25 10:30:00	2	pump and filter maintenance	Chelsea Jia	206-857-6698	9433 NE 146th Cir Bothell Washingto...
2020-02-26 08:45:00	2	pump and filter maintenance	Chelsea Jia	206-876-7395	244 4th Ave Kirkland Washington 98...

2. Often times, when we purchase something, we will need to look up the details of our purchase order as soon as possible. This stored procedure provides functionality to do just that. Furthermore, the procedure follows what a normal purchase order (PO) will appear with how much of each line item as the total, but without the total PO cost at the bottom.

/* Creating Stored Procedure to find PO details */

```
DELIMITER //
CREATE PROCEDURE specify_PO
(IN PurchaseOrderID int)
BEGIN
	select po.purchase_order_id "PO Number",
		po.purchase_order_date "Purchase Order Date",
		s.supplier_name "Supplier Name",
		p.product_name "Product Name",
		pod.purchase_price "Purchase Price",
		pod.quantity "Quantity",
		pod.purchase_price * pod.quantity "Line Amount"
	from ((PurchaseOrder po right join PurchaseOrderDetail pod on po.purchase_order_id =
pod.purchase_order_id)
		left join Product p on pod.product_id = p.product_id)
		left join Supplier s on po.supplier_id = s.supplier_id
	where po.purchase_order_id = PurchaseOrderID
	order by po.purchase_order_id;
END //
DELIMITER ;

CALL specify_PO(9);
```

Output:

PO Number	Purchase Order Date	Supplier Name	Product Name	Purchase Price	Quantity	Line Amount
9	2019-11-22 12:57:52	Wholesale Resort Accessories	Floating Basketball Hoop	18.50	3	55.50
9	2019-11-22 12:57:52	Wholesale Resort Accessories	Inflateable Ride-On	21.70	1	21.70
9	2019-11-22 12:57:52	Wholesale Resort Accessories	Inflateable Ride-On	27.70	2	55.40

3. We may want to know which customers buy from our company most frequently, so we can easily find the top customers by sales using the following stored procedure. This will be particularly important as we may consider giving some rewards or discounts to the top three or five customers.

/* Creating Stored Procedure to find TOP customers by sales */

```
DELIMITER //
CREATE PROCEDURE top_customer
(IN TopNo_ranking_by_costomer_sales int)
BEGIN
SELECT p.Customer_ID Customer_ID, c.first_name First_Name, c.last_name Last_Name,
sum(p.Payment_Amount) Total_Payment
FROM mm_cpsc5910team01.Payment p join mm_cpsc5910team01.Customer c
ON p.customer_id = c.customer_id
Group By customer_id
order by total_payment desc
limit TopNo_ranking_by_costomer_sales;
END //
DELIMITER ;

CALL top_customer(5);
```

Output:

Customer_ID	First_Name	Last_Name	Total_Payment
9	Stephanie	Campbell	7860.08
6	Walter	Cameron	4915.00
5	Joy	Macon	3973.70
4	Jeffrey	Gross	3684.50
3	Elaine	Mecham	2548.00

4. To acknowledge whether a contractor has done well for the business quantitatively, we need to examine how much business has been brought to our operations. Normally, the better and more timely their work is, the more they are recommended and sought after by our customer base. We can then know whether the contractors are worth keeping our business relationships with or not. This should also propel contractors to give more than satisfactory service to our clients.

   /* Find TOP contractors by sales */

```
select c.contractor_id "Contractor ID",
    concat(c.first_name, " ", c.last_name) "Contractor",
    sum(py.payment_amount) "Business brought"
from ((PastService ps inner join Payment py on ps.payment_id = py.payment_id)
        inner join Contractor c on ps.contractor_id = c.contractor_id)
group by c.contractor_id
order by sum(py.payment_amount) desc;
```

   Output:

Contractor ID	Contractor	Business brought
9	Orion Dutnall	7781.00
6	Hamil Zollner	4915.00
5	Jdavie Volkes	3973.70
4	Sherwin Mintram	3684.50
3	Beale Peto	2548.00
11	Dacey Leahy	1699.99
12	Deeyn Rubenfeld	1608.00
8	Blondie Gobell	953.00
10	Editha Stainton	708.00
1	Chelsea Jia	654.80
7	Felipa Demicoli	639.50
2	Darb Bateman	358.70

5. This query is to show the sales amount for each product from high to low so that we can investigate why some products may not be performing as well as others.

   /* Show sales amount for each product from high to low */

```
select sd.product_id, p.product_name, sum(sd.quantity)*p.sale_price Total_Sale
from mm_cpsc5910team01.Product p join mm_cpsc5910team01.SalesOrderDetail sd
on p.product_id = sd.product_id
group by sd.product_id
order by Total_sale desc;
```

   Output:

	product_id	product_name	Total_Sale
▶	1	SilkGuard Complete Tabs/Sticks	52.99
	6	Body Board	50.00
	3	Smart Shock	39.54
	2	Brominating Tablets	32.95
	4	Burnout 73	28.14

6. This query is to show the total profit of each product from high to low. This is a similar query to 5), but takes into account the price margins associated with each item so we can adjust prices if needed on better or worse performing items.

/* Show the profit for each product from high to low */

```
select sd.product_id, p.product_name, sum(sd.quantity)*(p.sale_price -
(select avg(pd.purchase_price) from mm_cpsc5910team01.PurchaseOrderDetail pd
group by pd.product_id
having pd.product_id = p.product_id)) Total_Profit
from mm_cpsc5910team01.Product p join mm_cpsc5910team01.SalesOrderDetail sd
on p.product_id = sd.product_id
group by sd.product_id
order by Total_Profit desc;
```

Output:

	product_id	product_name	Total_Profit
▶	1	SilkGuard Complete Tabs/Sticks	7.000000
	6	Body Board	6.600000
	3	Smart Shock	5.190000
	2	Brominating Tablets	4.400000
	4	Burnout 73	4.000000

7. This query aims to aggregate all the inventory outflow and inflow to check the current inventory level for certain products. This will be very helpful in identifying when to buy certain products and is very integral to the supply chain and operations of the business.

/* Show current inventory level */

```
select p.product_id,
        p.product_description "Product Description",
        p.quantity_in_stock "Base Stock",
        sum(pod.quantity) "Purchase Order Quantity (+)",
        sum(sod.quantity) "Sales Order Quantity (-)",
        (ifnull(p.quantity_in_stock,0) + ifnull(sum(pod.quantity),0) - ifnull(sum(sod.quantity),0)) "Current
Inventory Level"
from ((Product p left join SalesOrderDetail sod on p.product_id = sod.product_id)
        left join PurchaseOrderDetail pod on p.product_id = pod.product_id)
group by p.product_id
order by p.product_id;
```

Output:

product_id	Product Description	Base Stock	Purchase Order Quantity (+)	Sales Order Quantity (-)	Current Inventory Level
1	Sanitizer (8 lbs)	15	4	2	17
2	Sanitizer (5 lbs)	20	6	2	24
3	Shock (6 pack)	100	4	2	102
4	Shock (6 pack)	75	6	2	79
5	Algea killer (1 qt)	40	4	NULL	44
6	Swim board trainer	10	6	4	12
7	24 inches	10	7	NULL	17
8	Unicorn shape	5	7	NULL	12
9	Blue mesh	5	4	NULL	9
10	Gold pony	5	6	NULL	11

8. This query is to show the total purchase amount from each supplier so we know who we spend the most with. This can help us negotiate for lower prices and find cheaper alternatives if need be.

/* Show purchase amount from each supplier */

```
select po.supplier_id, s.supplier_name, sum(pd.purchase_price*pd.quantity) Total_Amount
from mm_cpsc5910team01.PurchaseOrder po join mm_cpsc5910team01.PurchaseOrderDetail pd
on po.purchase_order_id = pd.purchase_order_id join mm_cpsc5910team01.Supplier s
on po.supplier_id = s.supplier_id
group by po.supplier_id;
```

Output:

supplier_id	supplier_name	Total_Amount
1	Bioguard	333.81
2	Wholesale Resort Accessories	1072.69

9. The query is to show employees' performance both in terms of total transactions and total sales. Some items may be more expensive than others, which has a large impact on the total sales an individual employee will accrue. This will help take the quantity of items sold into account as well when analyzing employee performance.

/* Show employee performance from high to low by sales amount */

```
select s.employee_id, concat(se.first_name, se.last_name) Employee_Name, count(*) No_Of_Transaction,
sum(p.payment_amount) Sales_Amount
from mm_cpsc5910team01.SalesOrder s join mm_cpsc5910team01.SalesEmployee se
on s.employee_id = se.employee_id join mm_cpsc5910team01.Payment p
On s.payment_id = p.payment_id
group by se.employee_id
order by Sales_Amount desc;
```

Output:

employee_id	Employee_Name	No_Of_Transaction	Sales_Amount
3	Cheryl Evans	2	67.68
2	Janet Levy	1	52.99
6	Nancy Brunton	1	50.00
1	Gail Draughn	1	32.95

10. This query is to show the total sales across stores. We can then compare locations and start to investigate reasons as to why one of our stores does better than another and find solutions to that problem whether it is with marketing or product sales.

/* Show the total sales amount for each store */

```
select se.store_code, sum(p.payment_amount) Total_Sales_Amount
from mm_cpsc5910team01.SalesOrder so join mm_cpsc5910team01.SalesEmployee se
on so.employee_id = se.employee_id join mm_cpsc5910team01.Store s
on se.store_code= s.store_code join mm_cpsc5910team01.Payment p
on so.payment_id = p.payment_id
group by se.store_code;
```

Output:

store_code	Total_Sales_Amount
1	100.63
2	102.99

11. This query is to show the total payment amount by month including both services and products. This can help us to analyze expenses and make adjustments to spending if we are exceeding previous months as well as plan for the future.

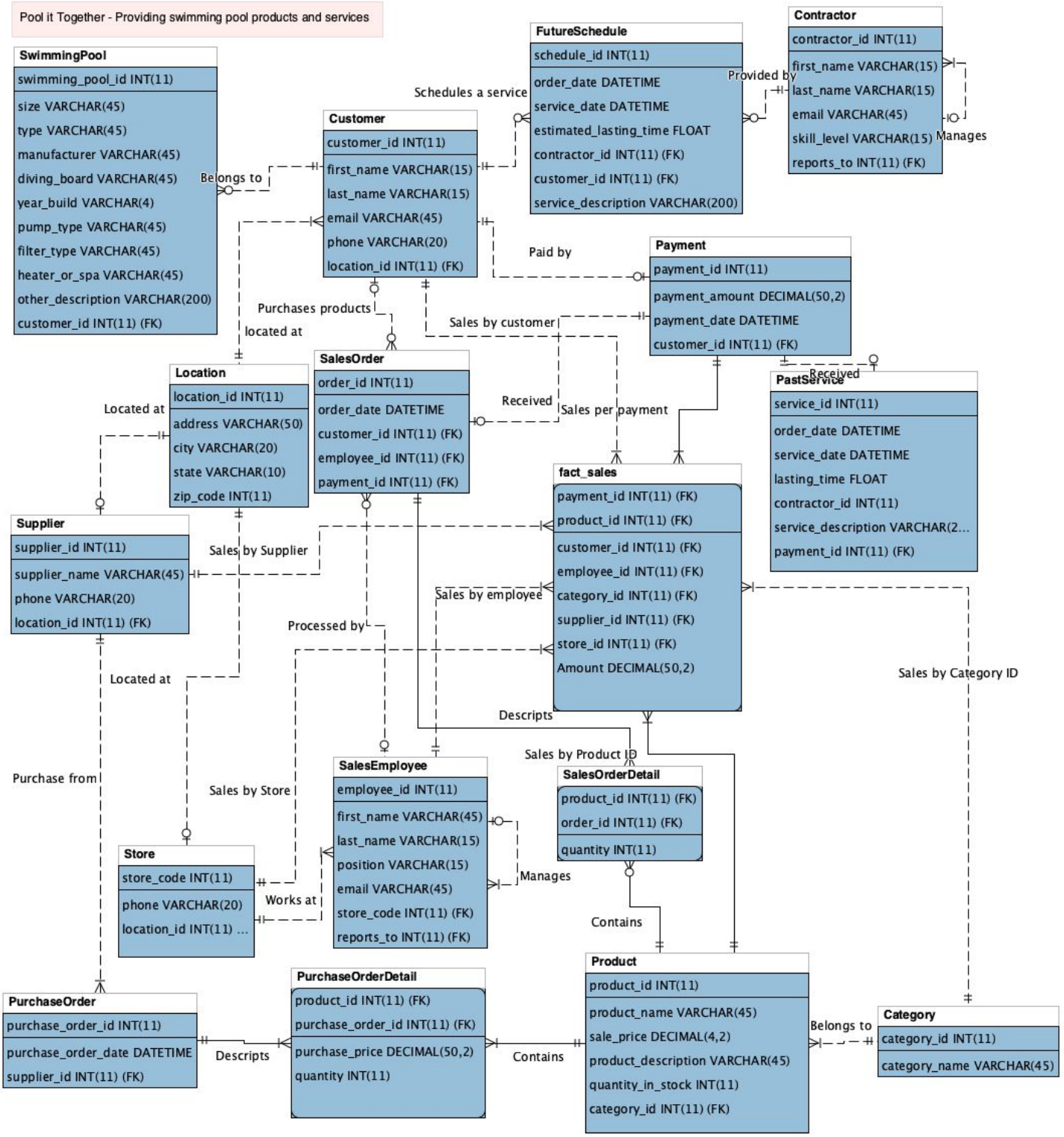
/*Show the total payment amount by month including both services and products*/

```
select extract(month from payment_date) as Month, sum(payment_amount) as payment_by_month
from payment
group by 1;
```

Output:

Month	payment_by_month
12	19471.38
11	639.50
1	9530.99
2	32.95

Physical Model (Fact table included)



Access MySQL database in the cloud

URL	**mysql-team1.cutaon14q9nk.us-west-2.rds.amazonaws.com**
Username	**team1**
Password	**team1isawesome**

Access Tableau Public

Tableau Public

Access Github Repository

mm_cpsc5910team01