CSD-310: Database Development and Use
Assignment 12.1: Case Study Critique
Blue Group: Bacchus Winery
Isaac Ellingson

Non-Technical Presentation Notes

This is the best looking presentation out of the bunch. The group introduction actually gave me a picture of who did what. We have business rules and python code here. I have everything I'd need to replicate your database and your process. On top of that, each report is textually connected to the client's needs with a "Use Case" statement.

My only nitpick here is just about the ERD presentation. Like us, you color-coded your ERD islands. If I didn't love that I wouldn't have done it myself. But somehow the gradients on the fields really grate on me.
I'm trying to understand why – I think part of it is a slight reduction in the contrast for reading the field names, especially on the pink gradients of ERD page 2 on slide 6. Part of it is the orange field which clashes with the pink and the green. And part of it is that it obscures the black outline of the entity.
Therefore, while I don't think you'd get a single point on your final grade for doing so, I feel it would improve the presentation to put the ERDs on a neutral background, remove the gradients in favor of a flat pale color, and ensure the entities are surrounded by a solid black line.

Again, stellar presentation, good job to everyone in the group.

ERD (The Technical Part)

Orders and Line Items

I think these entity and field names are going to cause great confusion in the future. For example, Supplier's primary key is SupplierID. SupplierDelivery's primary key is InvoiceID, suggesting that it may have been called Invoice in the past. SupplierItemDelivery's primary key is OrderItemID, suggesting that it may have been called OrderItem in the past.

Meanwhile, we've found even more different language for the "same" entities on the distributor end. DistOrder's primary key is OrderID, suggesting a table name of Orders. The table name DistItemOrderID sounds like a primary key for something, but it's a table and its primary key is OrderItemID.

Our team chose SupplyOrder and DistributorOrder to distinguish order types from each other, and SupplyOrderLine and DistributorOrderLine to represent line-items on each order. You don't have to choose the same names, but I feel that choosing consistent, symmetrical names for these tables and their primary keys will prevent a lot of confusion in the future.

Wine and WineInventory

While not mentioned in the case study, a winery would most definitely have different vintage years, and including this in the design was a good decision on behalf of the client.

Looking aside to the inventory, what does it mean to have 18 units of 1967 Cabernet and also 7 units of 1967 Cabernet? There does not appear to be a unique stocking location for each. I fear that as you look carefully at this entity to normalize it, what makes a WineInventory unique may be the *wineID*, so you could remove the WineInventoryID and make the WineID a FK+PK - - - but if everything in WineInventory depends on WineID in a 1:1 relationship, then we could skip the entity entirely and add a Quantity field in the Wine table.

If there's something useful about keeping multiple "pools" of the same wine available – for example, if you think you'll need to add a stock location, then you could do that instead and that relationship could be worth the extra query complexity and consistency burden.

Shipment, ShipService

Once again, it feels like an entity rename has occurred, and ShipService's PK is called ShipperID. But also, it's referred to as ShippingID from Shipment! Let's get everyone on the same page here, probably on ShipServiceID.

I find ShipperName / ShipperPhone / ShipperWeb to be a little weird too. Is this for natural join protection? Then why do WineInventory.Quantity and DistItemOrderID.Quantity have the same name? My recommendation, personally, is to drop table-named fields (ShipService.ShipperName → ShipService.Name) and pointedly avoid natural joins. But if your aim is to use them, be thorough and consistent, or else all you bought yourself is a false sense of security.

I think the relationship line from DistOrder.ShipmentID to Shipment.ShipmentID may be backwards. Based on the foreign keys present here, it appears that each order is sent entirely within a single shipment, but each shipment might represent the fulfillment of many orders.

Distributor Catalogues

Short note here on WineToDist. We felt that "whether a distributor carries a wine" was less of a formal, standalone piece of data, and more of an emergent property of distributor orders within a certain time period. But this may not be the case if there are explicit agreements with distributors to carry certain wines. In this case the many-to-many relationship depicted here is appropriate.

However, what does it mean for a distributor to carry the same year of the same wine three times? Consider using a composite ID on (WineID, DistID) instead of a standalone WD_ID to prevent these duplicates from being representable.

Hours, Departments and Management

Based on the fields laid out here, I'm not sure the 1:1 relationship from Department.ManagerEmployeeID to Employee.EmployeeID is correct. It would seem that one employee could manage many departments. The exception here could be if there's a UNIQUE constraint on the ManagerEmployeeID field, and I'm not sure that that would be a desirable change.

Once again, there is inconsistent primary key naming in Hours.PunchID.

This is not necessarily a problem, but it is possible for the same employee on the same day to punch three times with the same number of hours. This does not by itself represent an inconsistency and may actually be an intentional design choice. It is *also* likely possible to punch more than 24 hours in a day, and I don't think with this design there is any way you can stop that. So aside from reconsidering field names, I probably would not recommend any changes here.

Reports

Distribution Reports

For Wine by Distributor, I would expect to see distributors on the left, and wines on the right. This is a tiny nitpick- the report looks fantastic!

Sales by Wine looks great, I think I would prefer my top sellers on top though.

Wines that Haven't sold appears to be doing its job, although if we're looking at a report of wines that haven't sold anything, then the "TotalSold" column is superfluous.

Supplier Report

I worry that as deliveries become numerous that this report could spiral out of control. If tightly time-boxed, it could definitely be useful, but I might also like to see this information aggregated by supplier to make longer-term behaviors visible. I do really like the final computed column saying "Pending" / "On Time" / "Early" / "Late" at a glance.

Hours

The tuple array output in the first report is unusable. Reports are client-facing. We must live to a higher standard here.

The winery case study specifically asked about how employee hours trended over the last four quarters, and I don't see that question answered here. While these reports are useful, I would find it hard to meet their reporting requirements without running four time-boxed versions of these with a total line.

Python Code

While no part of the python code is required here, I do feel the need to mention that if you have your env file here named "setup.env" instead of ".env", then you probably want to also add it to your .gitignore so that this sensitive data can't get picked up by a commit and pushed to your public git repository. See https://git-scm.com/docs/gitignore which is the official docs, or https://www.w3schools.com/git/git_ignore.asp w3schools coming in clutch as usual with great user-friendly tutorials.

Assumptions

Extreme nitpick – "less issues" → "fewer issues" at the end.

Conclusion

This is a really strong presentation, and a good database design. The ERD display can be tidied up a bit, and some table and field names could be rethought. Finally, in a very small number of places we can double-check that our primary key is really the one we want.

These are some of the best looking reports in class, and I appreciate that each one comes with a Use Case statement to connect it directly with the client's needs.

A lot of what I've said in this review are extreme nitpicks. This project is in really good shape- I just held nothing back in this review. If there's a change I suggested here that just doesn't work for your team, trust your instincts. You'll do just fine.