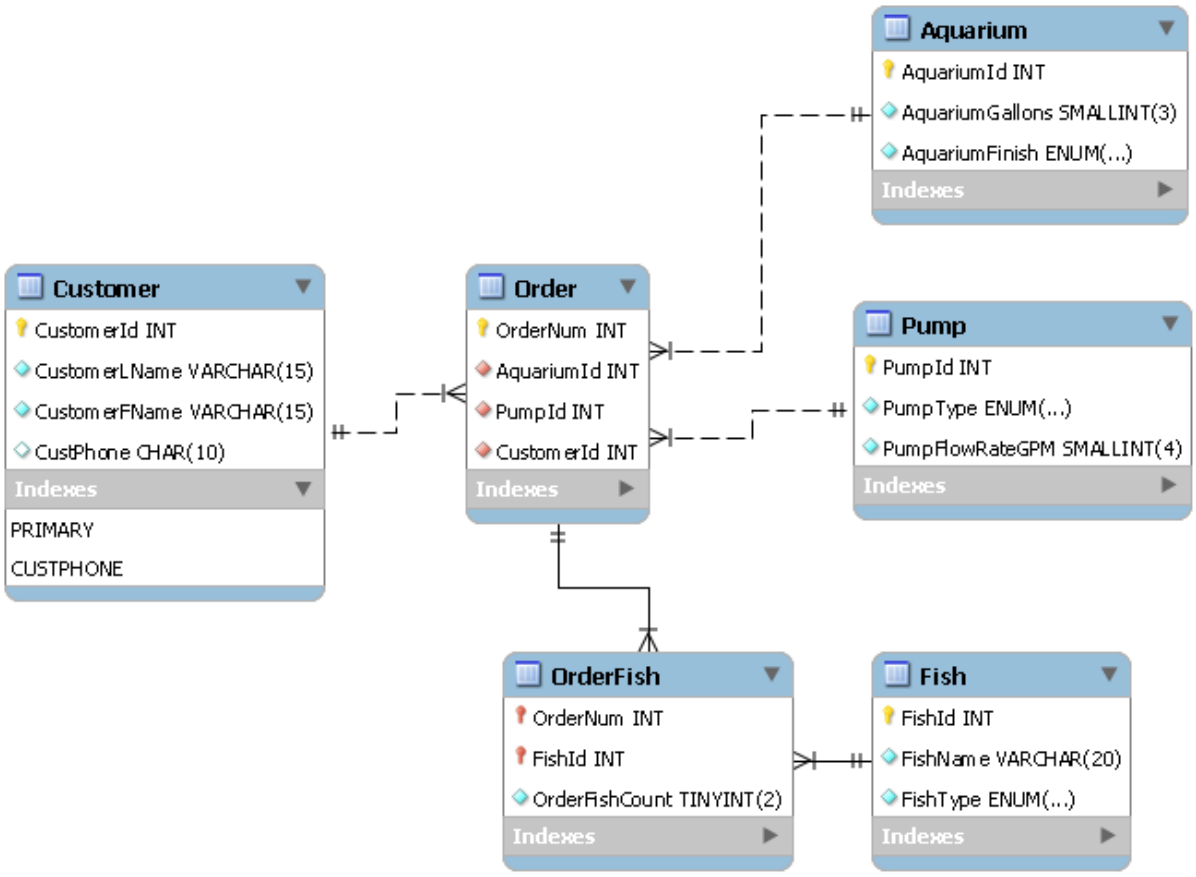
Project #6: Create a DB for Al's Aquariums

1 Al's Aquariums

You've been working with Al's Aquariums to design a database for their order system. You've learned that each aquarium order consists of one aquarium, one pump, and one or more types of fish to populate the tank. The database design you've come up with is as follows:



2 Create the Database

Write SQL to create the database and each of the tables. Each table should have the columns specified, along with the primary and foreign keys to support the relationships. Use auto-increment fields where there isn't a natural unique identifier to go on. Create the additional indexes shown (PK indexes are created automatically).

Create all PK, FK, and any additional Unique constraints *at the table level*, using this naming scheme: four letters representing *this* table's name, an underscore, the field name(s) in the key, an underscore, then an abbreviation indicating the type of constraint (PK, FK, UQ).

2.1 Enumerations

- AquariumFinish is either "Wood" or "Black"
- PumpType and FishType are both either "Salt" or "Fresh"

2.2 Deletion Plan

Implement a database-wide plan such that if any PK gets deleted (or updated such that orphans would be created), all corresponding FKs also get deleted.

3 Populate the Tables

Write additional SQL to populate sample records into all the tables. Supply enough data that you can do a few realistic queries to prove everything is working as expected. Note that FKs can't be automatically populated with realistic data; you'll need to generate data based on data you've put into other tables. Take some care in populating linking tables.

4 Write Test Queries

Write a few test queries that prove that all the expected joins are working, and that Al's will be able to get to, and put together, all the necessary data for their order system. Think of this as part of the proof of concept to show Al. These should be realistic, business-oriented queries. SELECT * doesn't count.

The best way to write these is to think of a business scenario *first*. Write that down in a SQL comment, then write a query that fulfills the business need.

5 Other Requirements

- Create these from scratch, by hand. Start with gated DROP instructions so each time you run your code it will wipe out what's there and start again.
- This should be *one* .SQL file, not a series of them. Ensure this runs as a *script*, meaning that with a single "run it all" command, the entire set of commands succeed. There should be no errors shown in the output window.
- One interesting way to test: reverse engineer an EER (ask!) and compare it to the one above.

6 Submitting Your Work

Submit your single SQL file in Canvas.

(continues…)

7 Grading

Grading on this project is about completeness. I'll consider these questions:

- Is a complete and working database created by your SQL?
- Is everything requested present?
- Are tables populated enough to prove the design works?
- Are test queries thorough enough to help with proof of concept to "sell" the idea for the database?

Area	**Percent**
Database drop/creation	5%
Table/Column creation	20%
PK creation	20%
FK creation & deletion plan	20%
Index creation	5%
Sample data	20%
Sample queries	10%
Total	**100%**