-- Tatiana Zihindula : SUPERVISOR ROLE

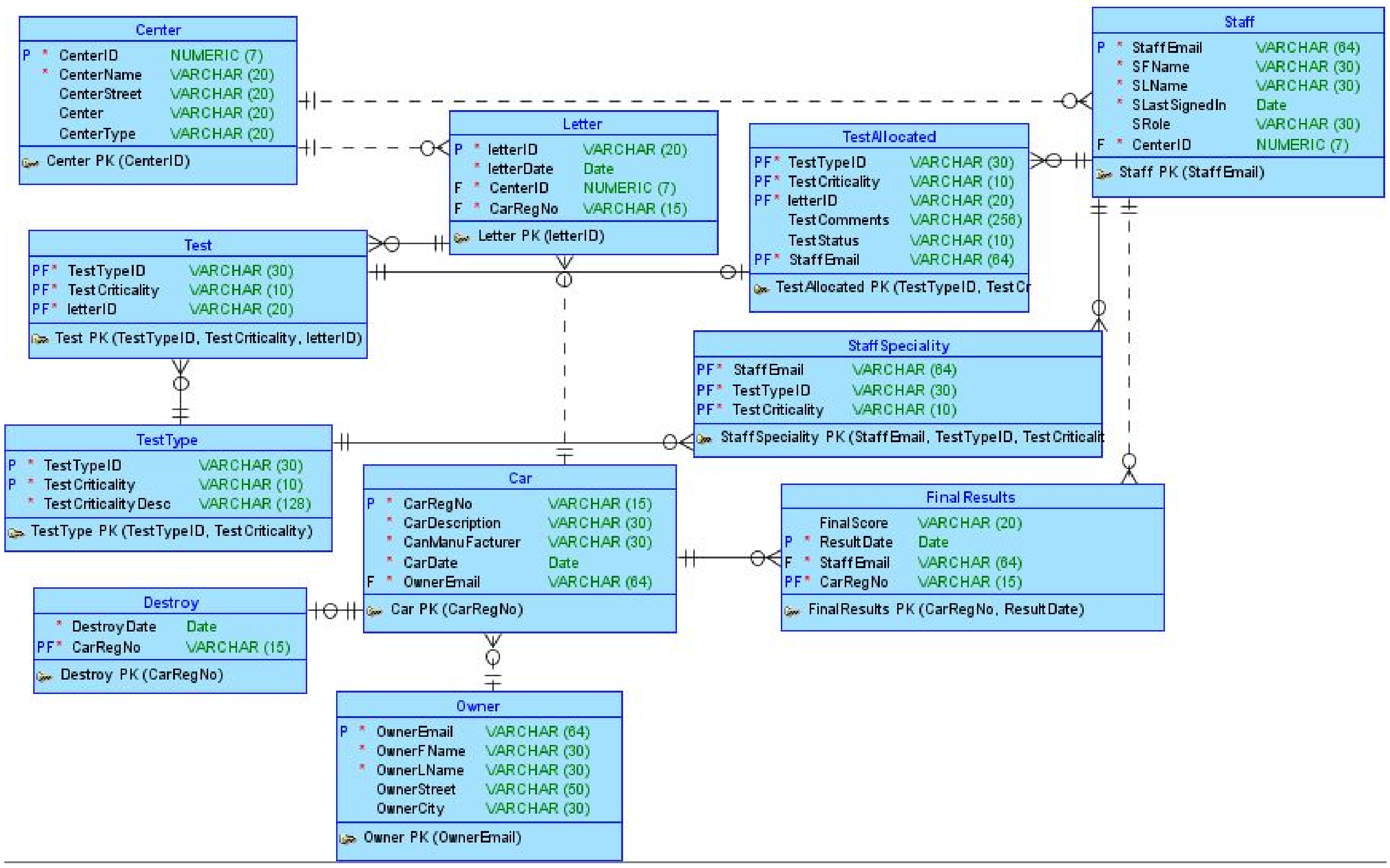
-- C16339923
-- CENTRAL CAR TEST ADMINISTRATION SYSTEM : Mid-Term Databases 2 CA

-- YEAR 3 :
This assignment is the implementation of the SUPERVISOR role from the Central Car case study.
This role depends on the **Mechanic** staff in test centers to provide comments on Tests they were allocated, on the **Owner** of the car to submit the letter received from the Central Car Center, and finally on the **Clerk** who oughts to send out letters.

I used **simulations** to highlight the dependencies of the supervisor role onto the cited roles, but I still developed this project as a stand-alone testable unit..

Project parts.

1. **Group Relational schema used**
   There are broken down entities to their third normal form to reduce redundancy or duplications or both.



Center
P * CenterID NUMERIC (7)
* CenterName VARCHAR (20)
CenterStreet VARCHAR (20)
Center VARCHAR (20)
CenterType VARCHAR (20)
Center PK (CenterID)
Letter
P * letterID VARCHAR (20)
* letterDate Date
F * CenterID NUMERIC (7)
F * CarRegNo VARCHAR (15)
Letter PK (letterID)
TestAllocated
PF* TestTypeID VARCHAR (30)
PF* TestCriticality VARCHAR (10)
PF* letterID VARCHAR (20)
TestComments VARCHAR (256)
TestStatus VARCHAR (10)
PF* StaffEmail VARCHAR (64)
TestAllocated PK (TestTypeID, TestCr
Staff
P * StaffEmail VARCHAR (64)
* SFName VARCHAR (30)
* SLName VARCHAR (30)
* SLastSignedIn Date
SRole VARCHAR (30)
F * CenterID NUMERIC (7)
Staff PK (StaffEmail)
Test
PF* TestTypeID VARCHAR (30)
PF* TestCriticality VARCHAR (10)
PF* letterID VARCHAR (20)
Test PK (TestTypeID, TestCriticality, letterID)
StaffSpeciality
PF* StaffEmail VARCHAR (64)
PF* TestTypeID VARCHAR (30)
PF* TestCriticality VARCHAR (10)
StaffSpeciality PK (StaffEmail, TestTypeID, TestCriticalit
TestType
P * TestTypeID VARCHAR (30)
P * TestCriticality VARCHAR (10)
* TestCriticalityDesc VARCHAR (128)
TestType PK (TestTypeID, TestCriticality)
Car
P * CarRegNo VARCHAR (15)
* CarDescription VARCHAR (30)
* CarManuFacturer VARCHAR (30)
* CarDate Date
F * OwnerEmail VARCHAR (64)
Car PK (CarRegNo)
Final Results
FinalScore VARCHAR (20)
P * ResultDate Date
F * StaffEmail VARCHAR (64)
PF* CarRegNo VARCHAR (15)
FinalResults PK (CarRegNo, ResultDate)
Destroy
* DestroyDate Date
PF* CarRegNo VARCHAR (15)
Destroy PK (CarRegNo)
Owner
P * OwnerEmail VARCHAR (64)
* OwnerFName VARCHAR (30)
* OwnerLName VARCHAR (30)
OwnerStreet VARCHAR (50)
OwnerCity VARCHAR (30)
Owner PK (OwnerEmail)

Thoughts when building the schema :

Too perfectly related but now no longer portable?

An issue we faced when implementing the design is to minimise the amount of foreign keys carried across tables. Though this tighteneses the relationships between tables it carries with it a lot of duplicated columns. For example, a car's registration number could be used as a car's primary key; but when a letter is sent, this letter becomes defined by its car. This car might have been tested before. So making the car's registration number the primary key of the letter ID could not work, because when the car is sent for the second time, the letter's primary key's UNIQUE constraint will be violated.

Hein?

Okay..
But the composition of the letter's date and the car's registration numbers are unique. Good. But then because the letter also defines a **specific Test on car at a point in time**, the composite key (letter date, car registration number) will now to be carried along with everywhere the tests on the car are needed!

Uhh!
Okay.. it's just in the Test table right? oh okay, in the TestAllocated table as well, which also has the staff's email
added to the bulk with the same problem, as well as the type of the test to do, which can also come in three different criticalities, whose type are not unique but the combination of the type and criticality are, okay? but, okayyyy now there are five foreign keys from composite keys being referenced in TestTypeAllocation? The staff staff emails we cannot with the car registration cannot be used alone as primary key to TestTypeAllocation since it won't be unique because one staff can specialise in many test types, and these test type already have types, for example, test type 'ENGINE', 'low' criticality differs from ENGINE,'high', So we better not reference the TestType anywhere else as it is already a composite key, okay, only In StaffSpeciality entity, okay, I am just going to add it to the Test table and That's it, Okay to the allocated table and I am done this time!!!!

~~Ugh! Let me just start coding, these things makes so much sense when you are actually writing some actual code..?~~
~~Huuu...~~
~~`Oracle (ORA-02270) : no matching unique or primary key for this column-list`~~
~~what??!- this should work, this should definitely work!!~~
~~This shou- this makes no sense..back to designing…~~

Okay, using sequences maybe? Maybe not? they do not show the real relationships between tightly related tables like these, and they are less likely to catch **parent-child** constraint violations and/or duplicated inserts.

Hmm, okay, what is it about the roles? staff members can be 'clerks', 'mechanics', 'supervisor' and these have to perform certain roles, the mechanic should not be allowed to allocate Tests to themselves for example and it's not to the supervisor to perform tests when a mechanic is absent!
Wait? is PL/SQL a OOP language? This sounds like an inheritance problem to me?!
~~Okay maybe create 3 tables? Clerk, Supervisor, Mechanic? then sele- No, this already sounds like a bad idea.. what if there are 10 more roles etc?~~
Oh there is also this concept of 'Test Center' and 'Central' Database? From which the clerk sends letters?

Hmm..
okay, maybe **using aggregation and roles to restrict** what each staff can do based on their role? Adding just a staff table with a column role that can take any value in the 3 roles, and adding another column to the center that can take any values in 'testcenter' or 'central' office can solve the inheritance problem.
What about private methods?
But that is OOP, this is not what RDS does?. Maybe **roles with restrictions are enough!**
If someone is a clerk for example, they could be restricted to view the mechanic that was allocated in a task.
Why? well I thought you'd' never ask! It's kind of none of their ~~business~~.. problem
Think about it;- . The president of the united state; well, Trump, going into the Cement's company database checking who is on duty for transporting the gravel needed to build the walls? Well, this might not break his rules but would would kinda break the law? This thing called data protection? ever heard of? no? Yeah, I thought you'd have, cause with the 'I agree' buttons on these forms we don't realise we are granting access to people from other roles to access.., but that's a different story..

back to the ERD.
Okay I mentioned

- Multiplicities many to many relationships were broken down into weak entities to cater repeated fields. e.g Mechanic and TestTypes
- The use of sequences were discouraged in tightly related table where duplicates need to be caught and parent-child relationship is important, e.g trying to submit a letter twice. a sequence could just increment, relationaly build primary keys will immediately catch this.
  ~~The inheritance problem was solved by adding a column in the parent field. eg. to Staff, add a role column where each specific person can enter who they are.~~
  Maybe this was not needed at all if granting restrictions are well build.
- For exclusive operation to specific columns such as updating availability of the staff, sending letters, testing cars granting restriction was used to restrict users access to tables that do not related to their roles. eg, the mechanic should only update the comment field and the status field in the allocated table. they should not change their ID to someone else's.
- **To fix the 'too many foreign composite keys Oh My God I can't breath'** problem, well, we didn't fix it! Because that's expected with relational databases. But for the letter ID, we made a group decision to concatenate the letter date and the car's registration number to allow it to be carried across, and minimise duplicated column.
- For the '**destroy**' car requirement, as a supervisor I made a decision of adding a Table called Destroy as opposed to delete the car from the system? where cars that are not road worthy could be dumped if they failed 3 times, as opposed to deleting them from the system.
  { This was inspired by a **real-life** scenario where the police once came to this Test Center investigating about a stolen car whose information was deleted a week ago, *but that's a different story!* }

2. **Group_schema.sql**
   The implementation of the ERD explained above
3. **Supervisor.sql**
   contains supervisor-related transaction.
   They are all functional in the group database and might not require recreating.
   They are ready to use with testcase.sql

4. **testcase.sql**
   Shows how Supervisor-related transactions are completed.
   it also stimulates the dependencies these transactions have to other roles such as relying on the clerk to send letter, the Owner to insert the letter and Mechanic to have updated these tables.

   How to use it?
   Run testcase.sql sequentially

   -> **The issue with this** simulation shows that as a supervisor I still have some columns granged to me that I should not be allowed to update.