Exercice 1
Dans la vue liste des ventes, afficher la dernière date de livraison ouverte. Si la date à plus de 12 mois alors afficher la ligne en rouge.

Solution:
N.B: For this exercise most of the solution is on the code itself, please refer to the custom module.
The idea here is to add a computed state field(selection) with 2 options and not visible ofcourse so when the delivery date is more than 12 months old the state should be **'late'** otherwise **'normal'** this way on the tree view I can decide when to make the delivery date visible and display it in red which is if the state is **'late'**.

Exercice 2
Ajouter un groupe utilisateurs "Sale Approver" dont le but est d'approuver les bons de commandes avant leur validation.
Un bouton "Approve" doit être ajouté sur le SO et n'est visible que pour le groupe d'utilisateur "Sale Approver"
Lorsque le bon de commande n'est pas encore approuvé il ne peut pas être confirmé, dans le cas où un utilisateur essaye de le confirmer il doit recevoir une erreur.
Lorsque l'on crée le SO on devrait pouvoir indiquer qu'il est à approuver, une activité doit être rajouté sur le SO indiquant à la sales personne qu'il doit approuver le SO.
Faire en sorte que l'on puisse filtrer facilement dans la liste de bons de commande pour afficher tous les bons de commandes qui doivent être approuvés.

Solution:
N.B: For this exercise most of the solution is on the code itself, please refer to the custom module.
The first part of the solution is to add a user group called **Sale Approver** and a button on the SO, but the button is only visible for the users with the group **Sale Approver**.
And the second part for the solution is to add a Boolean field to check whether the sale is approved or not. If approved then the user can proceed with the confirm button otherwise block the user from confirming the SO.
The third part is where I added a filter on the search view to display the SO that needs to be approved. I called it **'To be approved'**.
The final part of the solution is adding a todo activity for each new SO on creation, and make the approve button invisible when the SO is approved.
Note: Please note for this solution it can be adjustable to handle all the scenarios for SO, for example when the SO is confirmed and I want to cancel it and confirm it again...etc

Exercice 3: Requêtes SQL
Faire en sorte d'archiver toutes les vues d'un module spécifique à votre choix.
Créer une requête qui met le pays belgique sur tous les contacts ayant comme code postal, 1000, 5000 et 1300.

Solution:
To archive all views of a specific module I used the following query with the module **crm:**
“ update ir_ui_view set active=False where id in (select res_id from ir_model_data where module='crm'); ”

```
eeze=# update ir_ui_view set active=False where id in (select res_id from ir_model_data where module='crm');
UPDATE 261
eeze=#
```

And to set the country to Belgium for all contacts with postal codes 1000, 5000 and 1300, I used the following query:
“update res_partner set country_id=(select id from res_country where code='BE') where zip in ('1000','5000','1300');”

```
eeze=# update res_partner set country_id=(select id from res_country where code='BE') where zip in ('1000','5000','1300');
UPDATE 2
eeze=#
```

Exercice 4 : Analyse d’une solution

La Société X est une société de vente des services sur la plate-forme d’e-commerce Odoo.

Les prix des services, vendu en ligne, sont généralement vendus à faible prix (Exemple 50 dhs/ par commande).
Les gros clients effectuent beaucoup de commande la même journée et à chaque fois reçoivent un email de confirmation + un email qui contient les instructions de paiement pour payer par un virement bancaire et attendre leur paiement durant 2-3 jours. Par la suite la société X crée la facture, l'envoie au client et effectue la livraison.

Que penses-tu de cette procédure et qu’elles sont les conséquences sur la relation avec le client ?

Solution proposée :

La société X voulait améliorer cette procédure pour permettre seulement au gros clients la possibilité de payer un acompte (exemple : client paye acompte de 1.000 MAD) et que cet acompte soit transformé en crédits. Ces crédits seront utilisés pour payer des commandes futures, ceci permet aux clients de ne pas devoir faire un virement pour chaque commande.
Cela ne concerne pas les clients passagers

1- Q tu penses de cette solution ? est-ce qu'elle peut améliorer la situation, où vois-tu d'autres propositions ?

2- Si on retient cette solution, pourrais-tu réfléchir à la manière de l’implanter techniquement dans odoo? (Exemple : les champs à ajouter, les modèles, les interfaces à toucher, actions et les boutons à ajouter).

Note: Pas besoin de rentrer trop dans les détails, il ne faut pas implémenter la solution mais juste décrire le travail à accomplir.

Solution:
The proposed solution is a good idea because it makes things easier for big customers who buy a lot from Company X. Instead of paying for each order separately they can pay a deposit upfront which then gets turned into credits. These credits can be used for their future orders, saving them the hassle of making multiple payments.
However, there are a few considerations and potential improvements to make to ensure the effectiveness of the proposed solution:

- Clear communication with clients about the new payment process, make sure they understand how deposits work and how they can benefit from using credits for future orders.
- Consider offering the option for all clients to use the deposit and credit system if they wish this could appeal to smaller clients who also value convenience and could potentially increase adoption of the new payment method.
- Allow clients to choose the amount of deposit they want to pay, rather than having a fixed amount. This provides flexibility to the varying needs of different clients

To make this happen and implement it in Odoo a few things need to be done:

1. **Add a Field for Credit Balance**:
    - modify the customer model (`res.partner`) to include a new field for storing the credit balance.
    - this field should be of type float to accommodate decimal values representing the amount of credits
2. **update the Customer Form View**:
    - Add the new credit balance field to the customer form view so that users can see their current credit balance.
    - This can be done by modifying the XML file associated with the customer form view (`res.partner.form`) to include the new field.
3. **Modify the website Checkout Page**:
    - update the website's checkout page template to include options for paying a deposit or the full amount.
    - add a section on the checkout page to display the customer's current credit balance.
    - this involves modifying the HTML/CSS templates associated with the checkout page to include the necessary elements and functionality.
4. **Implement Deposit and Credit actions**:
    - Add buttons or links in the customer portal or account section for managing deposits and credits.
    - Create actions that allow users to view their credit balance, make deposits and use credits for orders.

    - these actions will trigger corresponding backend functions to update the credit balance and apply credits to orders.
5. **Automate credit Application to orders**:
    - implement server actions and triggers to automate the application of credits to orders.
    - Define rules that specify conditions under which credits should be applied to orders automatically.
    - for example, create a rule that applies credits to orders when they are confirmed based on the customer's available credit balance.