**14 JUIN 2018 - FORMATION DEV 450**

Id d'un objet : deux codes, soit le code 15 soit le code 18 (nombre de caractères).  
Le 15 est case sensitive, le 18 ne l'est pas. On compare ce qui est comparable : on ne peut pas comparer du 15 avec du 18.

L'apex renvoie du 18 ! On peut enlever les 3 derniers caractères pour récupérer du 15.

[admin@dev450.804125.com](mailto:admin@dev450.804125.com)

mdp : salesforce1salesforce1

**MODULE 5 - SOQL**

Tous les objets sont requêtables (99%, sauf certains très rares).  
Exécuté directement par la plateforme, permet **uniquement de faire de la requête**. Sinon on utilise le DML.

C'est un langage **gouverné**, qui empêche de faire du SELECT \* par exemple : on doit préciser les champs que l'on veut récupérer.

100 requêtes en une fois, 300 sous requêtes au total (SELECT imbriqués), remontée possible sur 5 niveaux, descente sur 1 niveau, 50.000 résultats renvoyés.

Intérêt : on peut interroger avec des account.name etc.

SELECT id, CaseNumber, etc  
FROM  
WHERE  
LIMIT  
ORDER BY

L'id est forcément récupéré, de façon implicite même si non déclaré.  
On doit définir lequel on veut récupérer, donc il **faut** utiliser un ORDER BY.

Une requête SOQL renvoie :  
- sObject  
- List<sObject>  
- Integer  
- Aggregat Result  
- Liste d'agregat Result

Where CloseDate != LAST\_N\_DAYS:365

Il est recommandé de toujours utiliser des listes pour récupérer les résultats de requêtes.  
Bonne syntaxe aussi : List <Case> cList = new List <Case> ([requête SOQL])

Schema : permet de récupérer les métadatas, en map.  
Schema.sObjectType.Case.getRecordTypeInfosByName().get('Product Support').getRecordTypeId();

* Permet de récupérer des objets de manière dynamique, sans utiliser de requêtes.

On est en revanche limité sur Schema.

Cela permet un code **dynamique**.

**MODULE 6**

Relation custom : \_\_r  
Relation standard

On peut **descendre à 1** niveau en SOQL, mais on peut **remonter à 5**.  
Eviter les 1-n, ça monte vite.

Limité à 300 sous-requêtes dans une requête.

Si l'on veut faire des requêtes avec jointures (externes ou internes), on sous requête avec du IN + sous requête.

Le FROM sur une relation, permet la jointure.

**MODULE 7 – DML ESSENTIALS**

DML pour la persistence.

Standalone DML 🡪 Si une erreur, terminé avec renvoie d'exception.  
Database.method(sObject List, false) 🡪 traitement partiel, AllorNone passé à faux.

Avec SalesForce les relations se font avec l'ID, il est donc important de savoir le manipuler.

**DEV 450 - Troisième journée**

Résumé de la deuxième journée : before update, after update, before undelete (mais pas le after), before insert, after insert,

Traverser du fils vers le père : si c'est un custom champs (relation), on peut remonter en prenant nomChamps\_\_r (r pour relation), on peut ainsi remonter de 5 niveau.

Pour descendre, on fait une sous-requête dans la clause SELECT ; on la fait sur la relation (nomObjetFils) au pluriel.

On peut descendre puis remonter (toujours 5 objets max pour la remontada).

Outer Join dans la clause WHERE pour vérifier que l'ID n'est pas présent (est présent, pour inner join).

On peut récupérer 50.000 éléments, on peut faire 300 sous requêtes, on peut faire 100 requêtes.

Requêtes dynamiques dans la version sans crochets (database.method()) de la requête.

Avant d'insérer, on doit vérifier :  
- la cohérence des données  
- l'existence de données  
- la non nullité des données (null = pas bon)

150 requêtes max dans une transaction, 10.000 max d'éléments.

Niveau trigger :  
On ne peut pas faire un insert ET un delete en même temps (même sur différents objets)

Trigger est bulkifié par défaut. En insérant 302 éléments, il y aura un pack de 200, puis un autre de 102. C'est différent d'une transaction.

PROF : Salesforce est une base Oracle, qui est commune pour tous.  
Salesforce a mis tous les records de tous les objets de tous les clients, dans UNE table Oracle (en vrai une 15aine, mais c'est l'idée).

Salesforce garanti un temps de réaction à moins de 300 ms !  
Comment gère t-elle ?

Avec l'ID de l'instance, l'ID de Company Profile.  
Chaque objet a un id, que l'on retrouve dans l'URL : a03 pour un objet custom.  
On peut faire jusqu'à 800 champs pour un objet. Un record fait 2 ko quoiqu'il arrive.

Personal account : compte + contact, qui fait donc 4 ko.  
*Taches : fait partie de Activity, séparé en tache & événement (hs).*

Il y a une autre table : la table des métadatas !  
Organisée sensiblement de la même façon, et enregistrant notamment les types.

Les champs formules sont interprétés à l'affichage.

Les trigger se déclenchent sur la table de datas. La formule n'ayant pas changée pour le jour, le trigger n'a pas été déclenché !

**Prof : si on fait un UPDATE, ça fait :**

[Voir ici clic](https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_triggers_order_of_execution.htm)

On fait un UPDATE (IHM), voici les étapes dans l'ordre :

Etapes :

1. D'abord, chargement du **old**

2. ...puis du **new**.

3. Déclenchement des validations système !

4. Déclenchement des règles de validation

5. Trigger : beforeUpdate, si plusieurs ils se déclenchent ... On ne sait pas quand. DONC BONNE PRATIQUE pas plusieurs trigger sur le même événement.

6. Validation systeme + Validation Rules

7. En base, on fait des calculs puis retourne l'id (permet la manipulation immédiate).

... autres règles que le prof nous épargne, puis on passe au

9. Tous les after update

... autres règles que le prof nous épargne, puis on passe au

11. Evaluation des workflow : il évalue la formule, et si elle est vraie il dit "on va faire des actions plus tard sur cela), ce qu'il fait en 12 (ordre qu'on ne connait pas)

12. Fait les actions (qui sont soit envoie d'un email, envoie d'un message SOAP, fieldupdate ou création d'un évé)

13. si Field Update (FU) 3+5+7+9

14. PB

15. Actions sur PB

16. si Field Update 3+5+7+9

... autres règles que le prof nous épargne, puis on passe au

18. Rollup

Si rollup, ça remet le bordel

CA c'est à maitriser. On fait des classes de test, pour gérer ça.  
On utilise des APEX uniquement si besoin.

La transaction commence au SAVE  
L'APEX Transaction commence étape 5 (All Before Update).

List, map, set : [clic](https://beginnersbook.com/2015/01/difference-between-list-set-and-map-in-java/)

**DEV 450 - Quatrième journée**

Résumé de la journée précédente

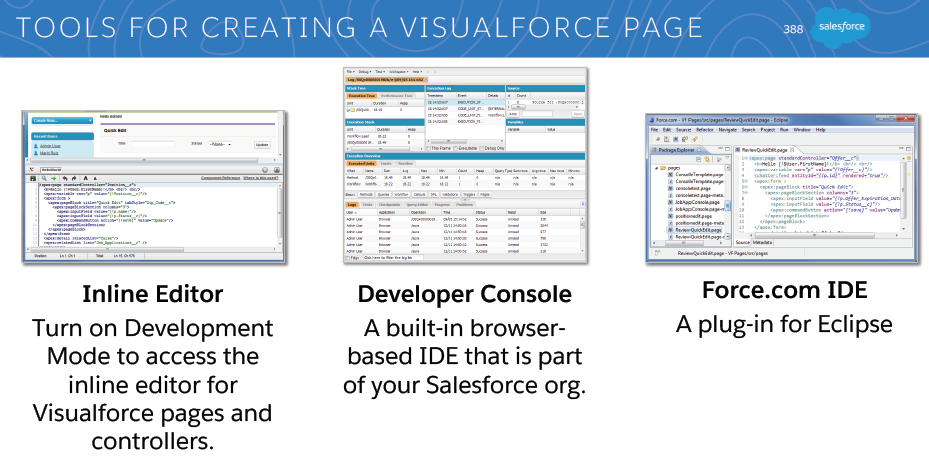
Tests : simuler ce que fait l'utilisateur, créer des méthodes privées, @TestSetup qui permet de lancer une création de datas qui se lance avant chaque méthode (déclarée par @isTest), startTest et StopTest pour réinitialiser les compteurs (remettre dans les conditions utilisateur).

On ne travaille pas sur la vrai base, rollback à la fin ; mauvaise méthode d'utiliser le paramètre de @isTest qui récupère les données de la base (SeeAllData = true)

Bonnes pratiques : jamais de DML SOQL dans boucles ou fonctions récursives, mettre des clauses WHERE pour récupérer uniquement ce dont on a besoin, utiliser les relations pour remonter les descendances et pas doubler les requêtes, FOR loop pour éviter de tuer la heap size, et dans un soucis de performance faire des clauses WHERE sur les champs indexés.  
Les trigger : on ne crée pas plusieurs trigger sur un événement, car on n'en maitrise jamais l'ordre. Corps du traitement trigger dans une classe (pour lisibilité et rendre factorisable un traitement).  
Les classes ne font que du traitement (pas obligatoire mais préférable, pour les triggers).  
On ne prend en paramètre que des listes car les triggers sont bulkifiés.

Penser aussi à vérifier qu'une appli n'est pas déjà développée dans le salesforce store répondant à notre besoin, avant d'envisager d'en créer une par nous-mêmes.

Création d'une page Visual Foce



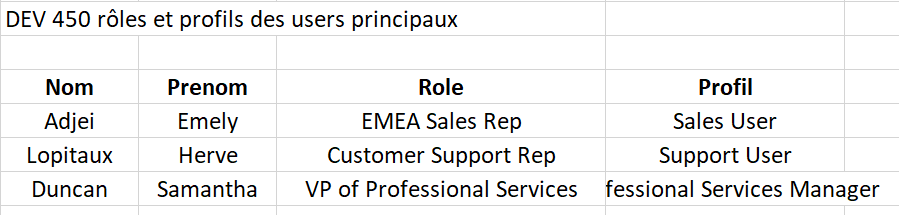
Professeur Abdel-Nor conseille la version Inline.

Adresse URL de l'édition : c.notreInstance.visualforce.com (c pour custom), car les serveurs d'hébergement de Salesforce ne sont pas forcément les mêmes.

Cocher développeur mode, puis mettre dans l'URL /apex/notreUrl (à voir en exercice)

**Reprise 14 juin – Formation autonome**

Environnement [alexis.masson-hdtf@force.com](mailto:alexis.masson-hdtf@force.com)

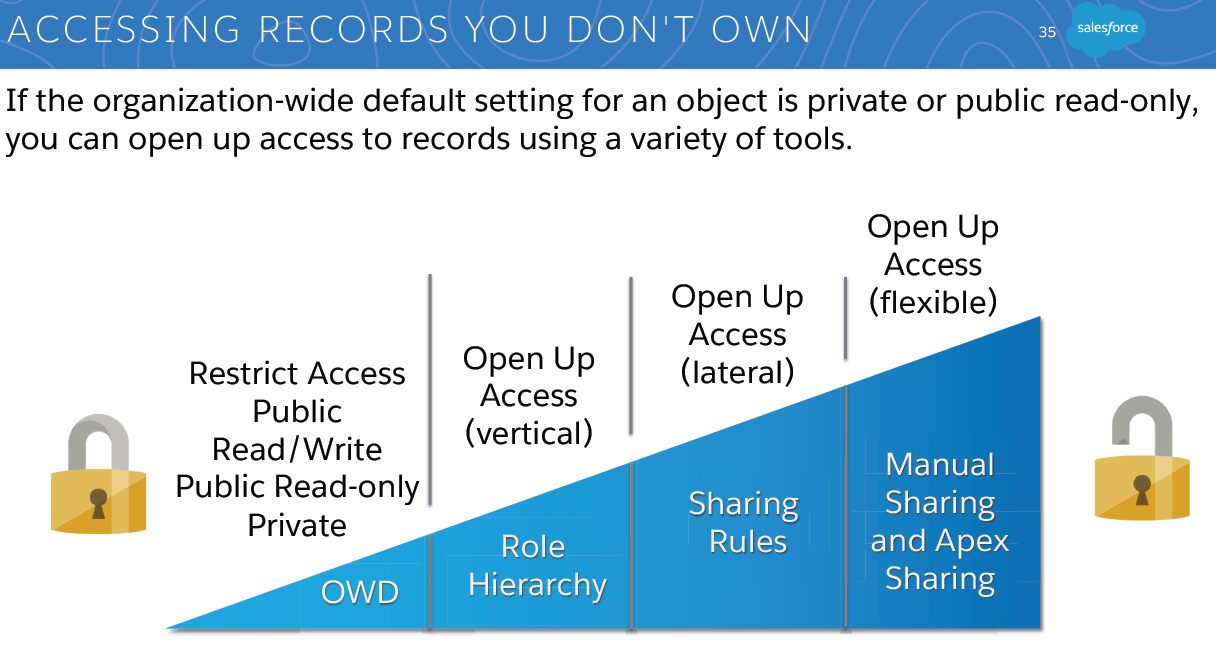


Licences Sandbox

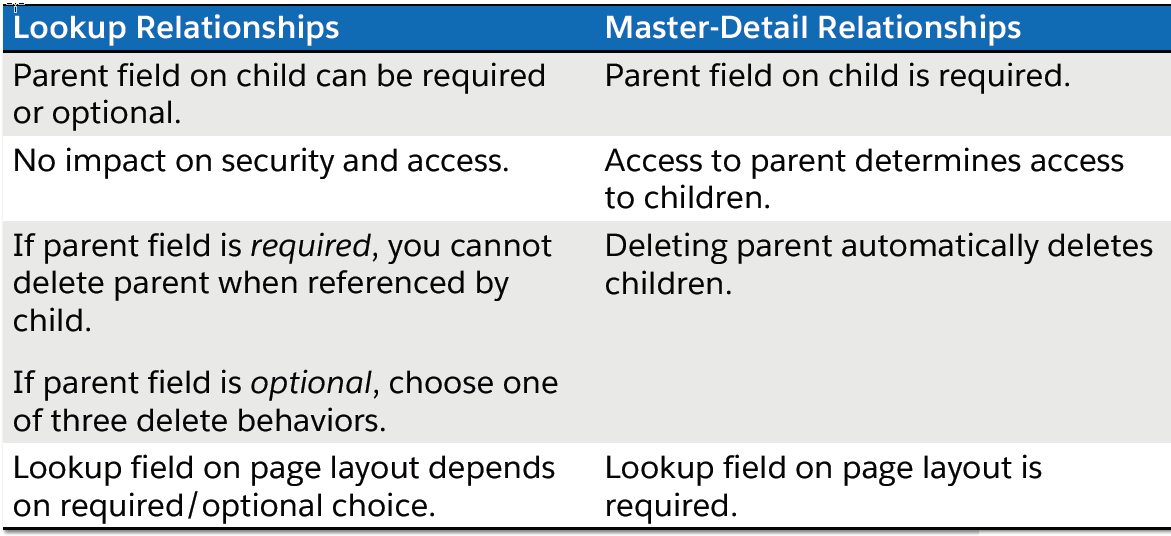


Opérateurs et fonctions : [clic](https://help.salesforce.com/articleView?id=customize_functions_i_z.htm&type=5)

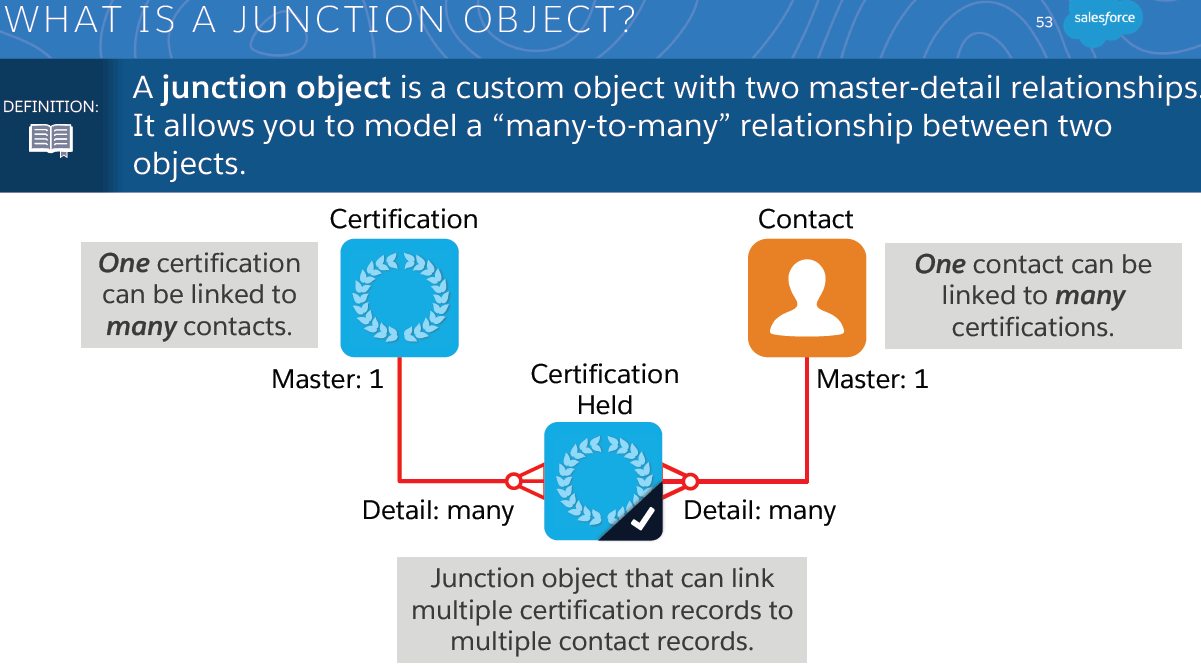
OWD



Lookup field vs Master-detail relationship

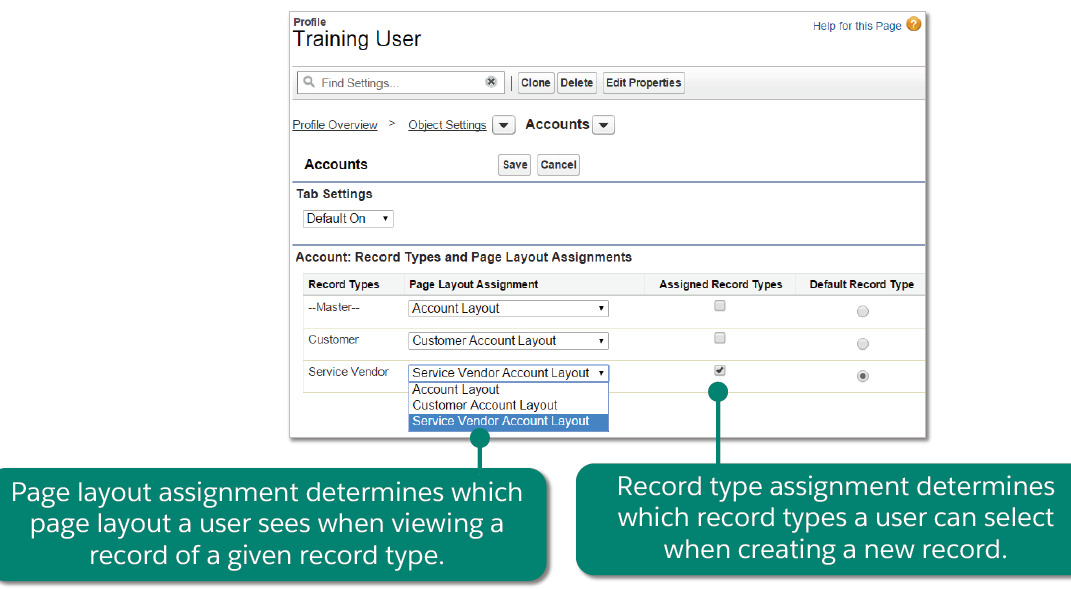


Many to Many \_\_r : junction object



Create lookup or master-detail relationships to model one-to-many relationships in SF.  
Use **jonction** **objects** to model **many-to-many** relationships.

Layout assignments

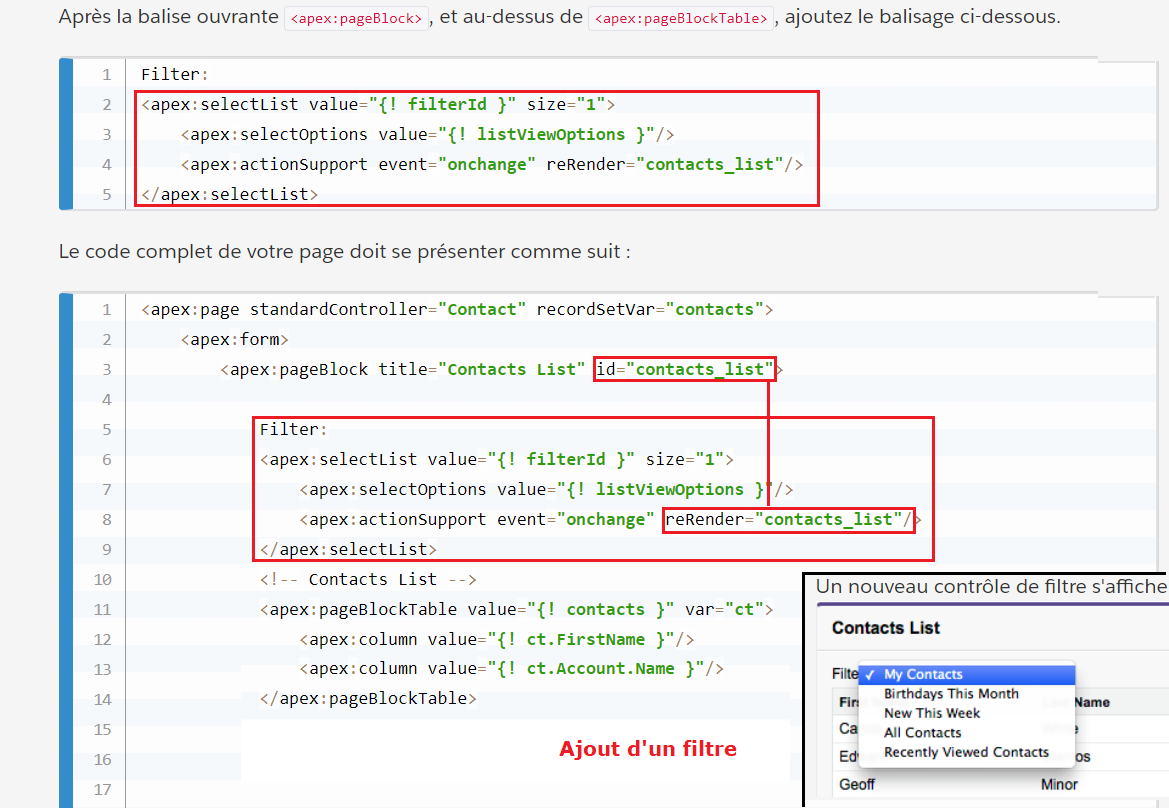


PageReference ! [clic](https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_system_pagereference.htm)

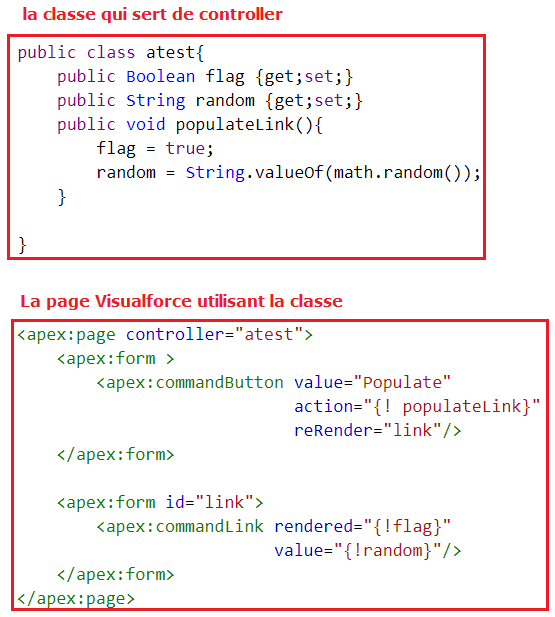
A propos des formules temporelles ! [clic](https://trailhead.salesforce.com/modules/advanced_formulas/units/date_formulas)

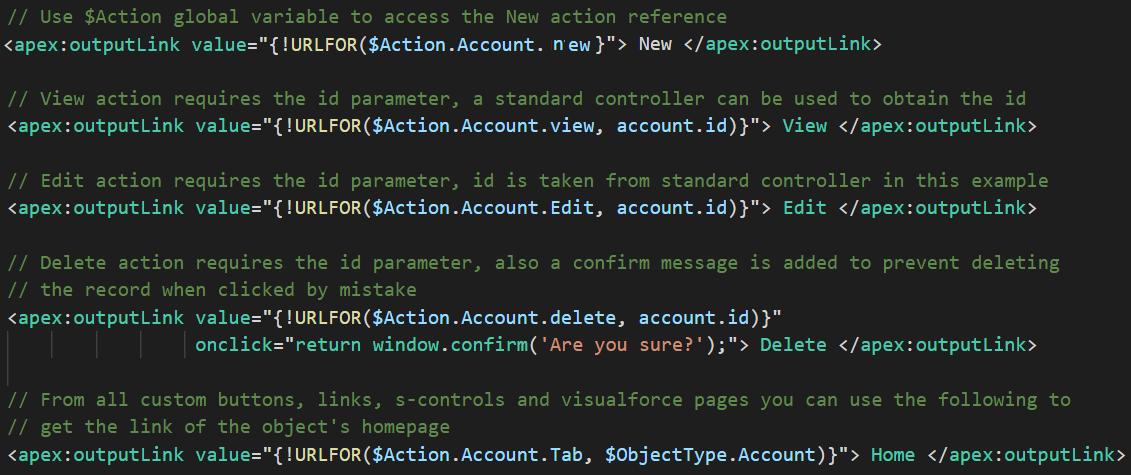
Récupérer la partie de gauche d'un texte (indice 0 du tableau splité) :

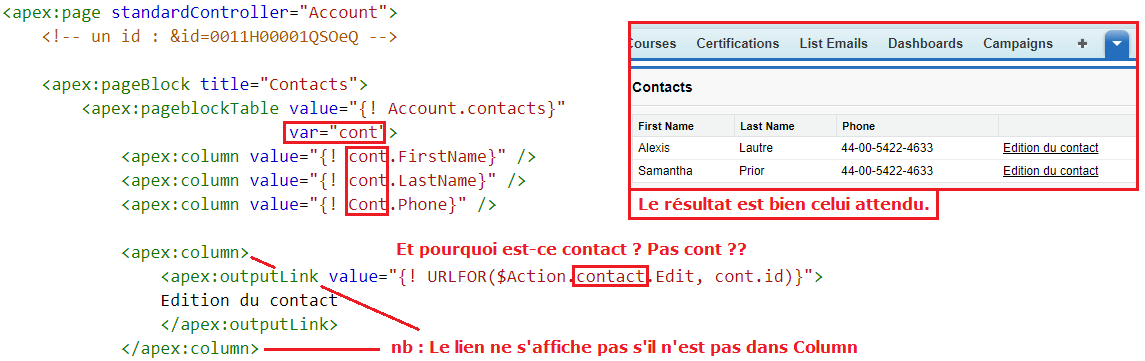
String solution=solutionPickVal.getLabel().split(',')[0];

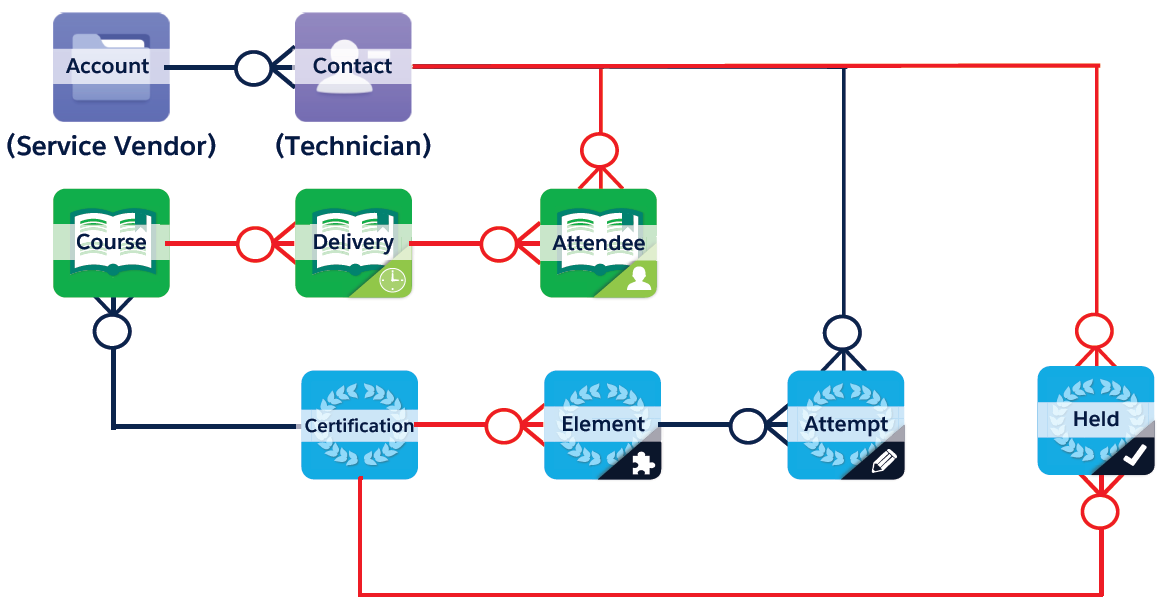


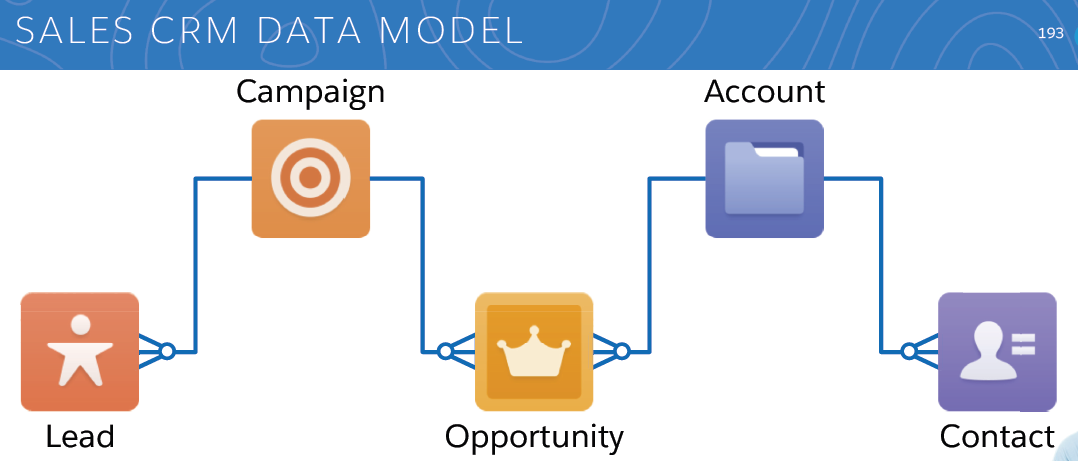












DEV 450 – Formation 2 Juillet 2018

805445

1 objet pèse toujours 2 ko (parfois 1 ko, ou 4ko, mais exceptions)

4 types de sandbox :

* Full : copie des datas de la prod (retrouve toutes les données de la prod)  
  *1 fois par 29 jours un refresh possible.*
* Partial : échantillon  
  *1 fois par 1 semaine (5j) un refresh possible.*
* Dev pro : aucune data, 1 Go  
  *1 fois par jour un refresh possible.*
* Dev : aucune data, et 200 Mo  
  *1 fois par jour un refresh possible.*

En France, c'est la Enterprise Edition la plus répandue (10 Dev + 1 Partial).

Tout est négociable dans SalesForce.

Il y a en commun toutes les métadatas (les objets custom) ; on a une copie des métadatas de la prod à l'instant de la création (ou quand on rafraichit la SandBox).

Quand on est en prod, les clients et contacts gardent les mêmes emails ! Certains autres emails sont changés (append d'un suffixe), sinon case à cocher pour envoyer vraiment les emails.

Sites conseillés par Abdel-Nor :  
[Salesforce Trust Status](https://status.salesforce.com/instances/EU10)  
[Developer salesforce](https://developer.salesforce.com/)

Une formule calculée sur une formule rollup, ne peut pas marcher (oula attention c’est du bordel mal pris).

Un champ formule ne peut prendre en compte que 10 autres relations, pas plus !

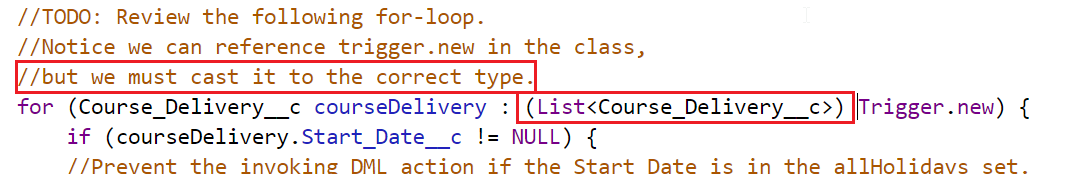
Un IF et un CASE ne consomment pas pareil : **utiliser des CASE** dans les champs formule.

On peut utiliser APEX dans :  
- pages visualForce  
- Web Service  
- email services  
- Lightning  
- Batch apex + scheduler  
- Javascript  
- site.com  
- API

Blob : les pièces jointes, documents etc, sont transformées en binaire et stockées dans un champ.

Les governor limits : [clic](https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_gov_limits.htm)

**MODULE 8 – TRIGGER**



**Droits et visibilité / lisibilité**

|  |  |
| --- | --- |
| Profile | Sharing model |
| * CRED / Objet * IHM (layout, RI) * Field Level Security : Cred au niveau du champs * System permission (droit d’accès à API ? Mot de passe qui expire ? Droit d’executer code Apex ?) * App permission (droit d’exécuter un Dashbord ? Droit sur Lead ? …) * Accès environnement : selon IP ? Selon horaire ? \_\_\_\_\_\_\_\_\_\_\_\_\_\_ * Permission set : pour éviter de cloner les profils, on crée un PS et on donne des droits supplémentaires à une personne précise (ne fait qu’ajout de droits, n’en retire pas) | * OWD par objet (public, privé, read/write) * Rôles : permet d’ouvrir, partage vertical * Sharing rules : partage horizontal * Apex ou manuel : on veut partager un record, par exemple départ en vacances je veux donner un record à un moment. |

La bonne pratique SF est de restreindre les droits, puis d’étendre petit à petit.

On peut voir un record, mais pas un champs (selon droits, ici FLS restrictif).

**Le bloc d’une transaction : toutes les opérations**

On part d’un edit d’un champs puis SAVE : ouverture de transaction.

1. Chargement du trigger.old
2. Chargement du trigger.new
3. Validation système : les champs obligatoires ont bien été renseignés, les champs uniques n’existent pas déjà en base, les types des champs sont bien respectés
4. Validation rules
5. Trigger : Before update ; on peut en avoir plusieurs donc déclenchement **des** triggers update. On ne connait pas l’ordre (d’où bonne pratique de rassembler)
6. A nouveau les validations système, puis les validation rules
7. On passe en base ! 🡪 opérations de base, pas de commit mais return un ID.
8. *(il se passe ensuite plein de choses standard à voir pour la certification)*
9. Trigger.after (tous les after)
10. *(il se passe ensuite plein de choses standard à voir pour la certification)*
11. Evaluation de tous les workflow
12. Action des workflow
13. Si field update 🡪 retour au 3 (validation system), puis 5 les Trigger.before (pas 4), puis 7 (passage en base), puis 9 avec les Trigger.after
14. Ensuite Process Builder, qui peut faire plein de choses DONT des field update !  
    Si field update, on refait 3, 5, 7, 9 d’affilé.
15. *(il se passe ensuite plein de choses standard à voir pour la certification)*
16. Rollup sur le père
17. Commit à la fin
18. Opérations post commit (ex : email envoyé par workflow ? On l’envoie maintenant)

La transaction commence au début, et termine à la fin ; on doit faire attention aux governor limits.

Workflow : une opération au clic, définition d’une règle, on va travailler sur tel objet, le workflow gère tel événement (insert, update), formule qui si elle est vrai on fait action : 4 types 🡪 message sauf xml, création d’une tache dans un calendrier utilisateur, ou **maj d’un champ**

**Conseil prof :** faire des classes de tests qui simulent le cycle : travail long et fastidieux, pour pouvoir affronter les éventualités de l’avenir (maj, ajout de process, de workflow),  
Il suffit ensuite de run les tests à chaque MAJ.