Salma Muhammad

ACCUEIL
TECHNIQUES DE BASES
Interface Front
Applications Back
Créer la base de données
Manipuler la base de tests
Interroger la base de données
Fiche révisions BDD
Évaluation BDD
Découverte de PHP
Publier le site Web
Évaluation PHP
PERIODE EN ENTREPRISE

Je déclare avoir terminé cette séance...

MySQL - Créer et administrer la base de données

Lexique

LDD = Language de definition des données qui sert à la création de **BDD** (Base de données) physique dans un SGBDr.

SGBDr = le modèle relationnel (Système de gestion de bases de données relationnelles), les données sont enregistrées dans des tableaux à deux dimensions (lignes et colonnes). La manipulation de ces données se fait selon la théorie mathématique des relations. exemple de SGBDR : Microsoft Access, Microsoft SQL Server, MySQL, Oracle, MariDB (fork de MySQl) etc...

L'environement de developpement

LARAGON

Laragon est un environnement de développement Web dédié au système d'exploitation Windows.

Il comprend trois serveurs : • NGINX / Apache (serveur web) • PHP (langage interprété côté serveur) • MySQL (base de données)

Et des outils pour l'administration des BDD : • HeidiSQL • PhpMyAdmin

EASY PHP

EasyPHP est un environnement (de type WAMP = Windows + Apache + Mysql + PHP= plate forme de developpement web) permettant de faire fonctionner localement (sans se connecter à un serveur externe) des scripts PHP.

Il comprend : • un serveur web Apache • un serveur de BDD MySQL • un moteur d'interprétation du langage PHP (script) • un outil d'administration de bases de données MySQL : phpMyAdmin.(peut gérer 1 ou n BDD) Il permet donc d'installer en une seule fois tout le nécessaire au développement local du PHP.

Ces logiciels gratuits sont principalement utiliser pour tester un site Internet en local (hors hébergement) avant sa mise en production sur un véritable hébergement Internet

Execution d'un script SQL

```
DROP DATABASE IF EXISTS *nom_de_la_base*;

CREATE DATABASE *nom_de_la_base*;

USE *nom_de_la_base*;

CREATE TABLE *nom_de_la_table_1* (
    champ_1 type(x),
    champ_2 type(x)
);

CREATE TABLE *nom_de_la_table_1* (
    champ_1 type(x),
    champ_2 type(x)
);
```

PRIMARY KEY

```
champ_id INT PRIMARY KEY NOT NULL AUTO_INCREMENT
```

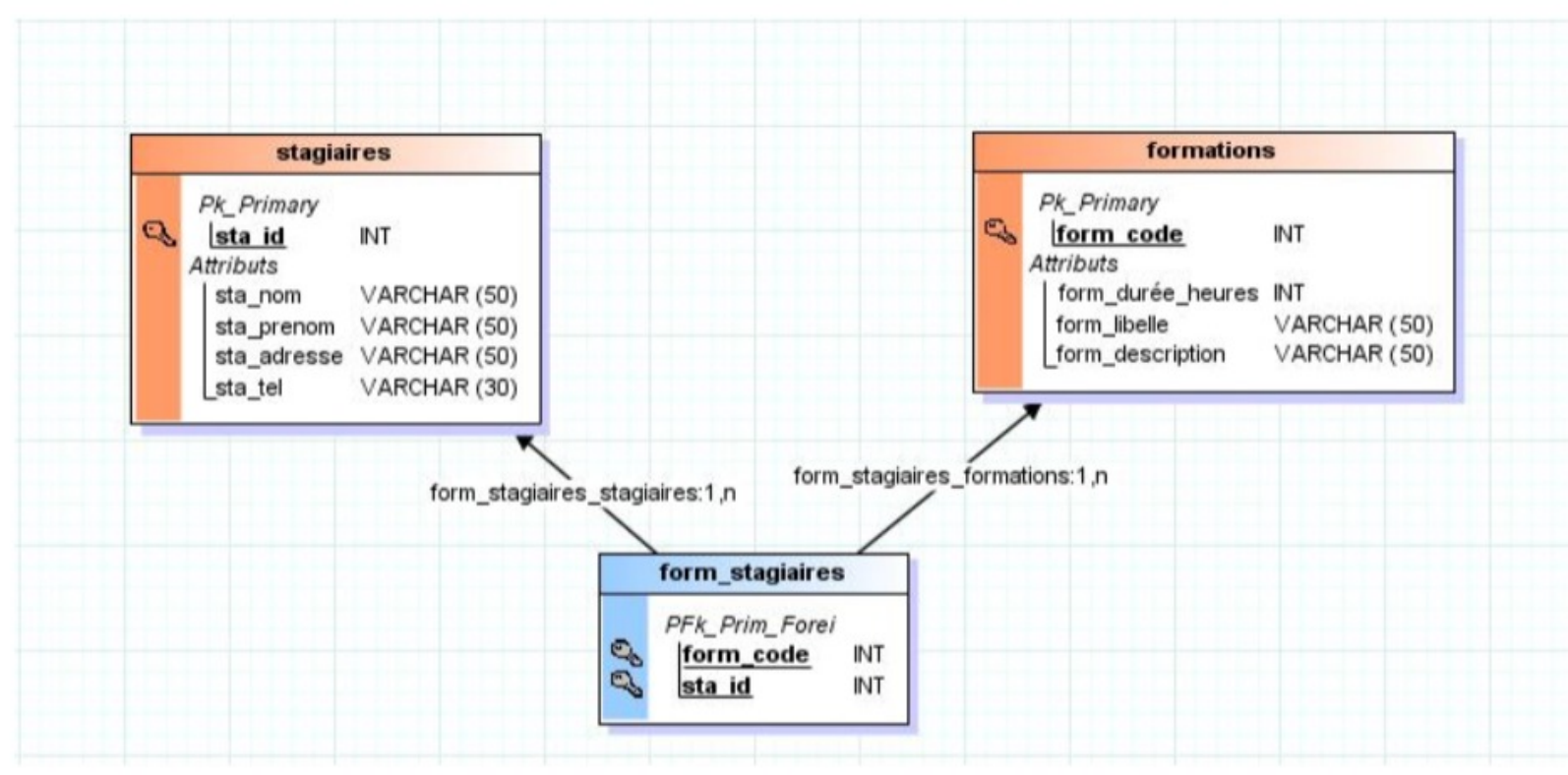
Elle permet d'identifier chaque enregistrement dans une table de base de données. Chaque enregistrements de cette clé primaire est composée d'une ou plusieurs colonnes. L'usage le plus fréquent consiste à créer une colonne numérique qui s'auto increment à chaque enregistrement.

FOREIGN KEY

```
autre_id INT,
FOREIGN KEY (autre_id) REFERENCES *nom_de_la_table_1*(autre_id)
```

La clé etrangére permet de relier deux tables entre elles. C'est un champ faisant réference à la clé primaire de l'autre table.

Exemple d'un Modele Conceptuel des Données et de son script MySQL :



```
CREATE TABLE stagiaires(
        sta_id      Int NOT NULL ,
        sta_nom     Varchar (50) NOT NULL ,
        sta_prenom  Varchar (50) NOT NULL ,
        sta_adresse Varchar (50) NOT NULL ,
        sta_tel     Varchar (30) NOT NULL
    ,CONSTRAINT stagiaires_PK PRIMARY KEY (sta_id)
);

CREATE TABLE formations(
        form_code         Int NOT NULL ,
        form_duree_heures Int NOT NULL ,
        form_libelle      Varchar (50) NOT NULL ,
        form_description  Varchar (50) NOT NULL
    ,CONSTRAINT formations_PK PRIMARY KEY (form_code)
);

CREATE TABLE form_stagiaires(
        form_code Int NOT NULL ,
        sta_id    Int NOT NULL
    ,CONSTRAINT form_stagiaires_PK PRIMARY KEY (form_code,sta_id)
    ,CONSTRAINT form_stagiaires_formations_FK FOREIGN KEY (form_code) REFERENCES formations(form_code)
    ,CONSTRAINT form_stagiaires_stagiaires_FK FOREIGN KEY (sta_id) REFERENCES stagiaires(sta_id)
);
```

INDEX

```
CREATE INDEX index_name ON *nom_de_la_table*(champ_1,champ_2,...);
SHOW INDEX from *nom_de_la_table*
```

La déclaration d'un index est utilisé pour la recherche dans une table, recuperer des données dans une DATABASE (BDD), l'utilisateur ne voit pas l'index, ils sont utilisés pour accélerer la recherche et les requêtes.

Création et suppression d'un utilisateur

```
CREATE USER 'nom_utilisateur'@'adresse_ip(ou %)' IDENTIFIED BY 'mot de passe';
DROP USER 'nom_utilisateur'@'adresse_ip(ou %)';
```

Utilisateurs et droits

CREATION DES PRIVILEGES

```
GRANT ALL PRIVILEGES ON nom_de_la_base.* TO 'utilisateur'@'adresse_ip'
IDENTIFIED BY 'mot_de_passe';
```

AFFECTATION DES DROITS

```
GRANT privilege
ON objet
TO utilisateur;
```

• privilege = SELECT, INSERT, DELETE, UPDATE, CREATE, DROP... • objet = nom_base.nomtable (exemple: papyrus.fournis) • utilisateur = nom de l'utilisateur

Exemple

```
GRANT SELECT, INSERT, UPDATE
ON papyrus.vente
TO dave_loper
IDENTIFIED BY '1Ksable';
```

Manipuler la base de données grâce au LMD

Pour modifier les données d'une base, l'utilisateur dispose de 3 ordres SQL :

- `INSERT`
- `UPDATE`
- `DELETE`

INSERT L'ajout de lignes dans une table (ou une vue) répond à la syntaxe suivante :

```
INSERT INTO NOM_DE_TABLE (NOMS DE COLONNES)
VALUES (LISTE DE VALEURS)
```

Le mode principal dans l'ordre `INSERT` pour ajouter des lignes est l'insertion directe avec la clause VALUES

L'attribution de valeurs est faite aux colonnes. *Exemples*

```
EMPLOYES et DEPART de structure
EMPLOYES (NOEMP, NOM, PRENOM, DEPT, SALAIRE)
DEPART (NODEPT, NOMDEPT)
```

Insérer l'employé *00140*, de nom *REEVES*, de prénom *HUBERT* dans le département *A00*, de salaire 2100€ :

```
INSERT INTO employes (NOEMP, NOM, PRENOM, DEPT, SALAIRE)
VALUES (00140, 'REEVES', 'HUBERT', 'A00', 2100)
```

On donne une valeur pour chacun des attributs spécifiés dans l'ordre `INSERT` ; les valeurs de la clause `VALUES` doivent correspondre avec la liste des colonnes, les attributs non spécifiés prennent la valeur `NULL`.

Insérer l'employé *00140*, de nom *REEVES*, de prénom *HUBERT* dans le département *A00*

```
INSERT INTO employes (NOEMP, NOM, PRENOM, DEPT)
VALUES (00140,'REEVES','HUBERT','A00')
```

La colonne *salaire* prendra la valeur `NULL` pour cette ligne. Si cette colonne n'a pas été spécifiée comme pouvant être nulle, une erreur sera générée. La liste des colonnes peut être omise à condition que l'ordre d'insertion concerne toutes les colonnes de la table.

Nous pouvons insérer plusieurs lignes dans la table (avec un seul `VALUES`) :

```
INSERT INTO employes (NOEMP, NOM, PRENOM, DEPT, SALAIRE)
VALUES  (00140,'REEVES','HUBERT','A00', 2100),
        (00150,'JACQUARD','ALBERT','B00', 1800),
        (00999,'LOPER", 'DAVE', 'C00', 900)

```

Une colonne ayant une propriété `AUTO_INCREMENT` ne fait pas partie de la liste des colonnes : On peut toutefois forcer sa valeur en spécifiant la colonne.

UPDATE

L'ordre `UPDATE` est utilisé pour modifier des lignes de tables existantes et est composé de trois clauses :

```
UPDATE *NOM_DE_TABLE*
SET *NOM_COLONNE_1* = *VALEUR_1* [,… *NOM_COLONNE_n* = *VALEUR_n*]
WHERE *condition*
```

- `SET` : Nom des colonnes et leurs valeurs ou expressions mises à jour.
- `WHERE` : Critère de sélection pour la mise à jour d'une ligne (optionnel)

Exemple 1 : Modifier le salaire de l'employé *LOPER*, qui gagne désormais 1000 € (au lieu de 900 €):

```
UPDATE employes
SET salaire = 1000
WHERE nom = 'LOPER'
```

Exemple 2 : Modifier plusieurs valeurs d'un coup (on modifie les nom, prénom et adresse de l'employé 3) :

```
UPDATE employes
SET nom = 'LOPER',
    prenom = 'Dave',
    adresse = '15 avenue Tella'
WHERE noemp = '00999'
```

Exemple 3 : Augmenter le salaire de 20% de tous les employés

```
UPDATE employes
SET salaire = salaire * 1.2
```

> Attention, l'absence de clause `WHERE` impacte donc tous les enregistrements de la table

Exemple 4 : Augmenter le salaire de 20% de l'employé de matricule *00040*.

```
UPDATE employes
SET salaire = salaire * 1.2
WHERE noemp = 00040
```

Exemple 4 : Modifier le salaire (augmentation de 20%) de l'employé de matricule *00040*, et son affectation dans le service *A40* :

```
UPDATE employes
SET salaire = salaire * 1.2,
    dept = 'A40'
WHERE noemp = 00040
```

DELETE

L'ordre `DELETE` utilise trois clauses pour supprimer une ou plusieurs lignes d'une table.

```
DELETE [FROM] *NOM_DE_TABLE*
WHERE *CLAUSE*
```

- `FROM` spécifie le nom de la table ou les lignes seront supprimées
- `WHERE` : spécifie le(s) critère(s) de sélection (optionnel)

> Attention, en l'absence de clause `WHERE` , toutes les lignes seront supprimées (table vide).

Exemple 1 : Supprimer **tous** les employés de la table *EMPLOYES* :

```
DELETE FROM employes
```

Exemple 2 : Supprimer les employés du département *E21* :

```
DELETE FROM employes
WHERE nodept = 'E21'
```

Exemple 3 : Supprimer les employés du département *E21* qui habitent Amiens :

```
DELETE FROM employes
WHERE nodept = 'E21'
AND ville = 'Amiens'
```

FORMALISER DES REQUÊTES SQL

- `SELECT`
- `FROM`
- `WHERE`
- `ORDER BY`
- `GROUP BY`
- `HAVING`
- `JOIN`

La Syntaxe

```
SELECT **NOMS DE COLONNES OU EXPRESSIONS**
FROM **NOMS DE TABLES**
WHERE **CONDITIONS DE RECHERCHE**
GROUP BY **NOMS DE COLONNE DU SELECT**
HAVING **CONDITION DE RECHERCHE**
ORDER BY **NOM OU POSITION DE COLONNE DANS L'ORDRE SELECT**
```

Exemples : *Lister le contenu de la table EMPLOYES*

```
SELECT * FROM EMPLOYES
```

Lister le nom et le salaire des employés de la table EMPLOYES

```
SELECT NOM, SALAIRE FROM EMPLOYES
```

- `ALL`
- `DISTINCT`

L'option `ALL` est prise par défaut, toutes les lignes sélectées figurent dans le résultat. L'option `DISTINCT` permet de ne conserver qu'un exemplaire de chaque ligne en double

```
SELECT [ALL] nom col1 FROM nomtable
```

par opposition à

```
SELECT DISTINCT nomcol1 FROM nomtable
```

`SELECT` CONDITIONNELLE AVEC `WHERE`

La clause WHERE permet de préciser les conditions de recherche sur les lignes de la table.

Conditions de recherche de la clause WHERE : Pour spécifier la condition de recherche dans la clause WHERE, on utilise indifféremment l'un des opérateurs conditionnels ci-après :

Description	Opérateurs conditionnels
Opérateurs de comparaison	=, <>, !=, >, >=, !>, <, <=, !<
Comparaisons de plage	`BETWEEN` et `NOT BETWEEN`
Comparaisons de listes	`IN` et `NOT IN`
Comparaisons de chaîne de caractères	`LIKE` et `NOT LIKE`
Valeurs inconnues	`IS NULL` et `IS NOT NULL`

Exemple avec `WHERE` et `OR` : *Rechercher dans la table EMPLOYES, les données concernant les employés qui travaillent dans le départements A00 ou E01*

```
SELECT * FROM EMPLOYES WHERE WDEPT = 'A00' OR WDEPT = 'E01'
```

LES FONCTIONS DU `SELECT`

La clause `ORDER BY`

La clause ORDER BY permet de préciser une séquence de tri pour le résultat d'une requête. • `ASC` séquence croissante (valeur par défaut) • `DESC` séquence décroissante

Exemple : *Lister le contenu de la table employé, trié par département croissant et nom décroissant*

```
SELECT *FROM EMPLOYES
ORDER BY WDEPT ASC, NOM DESC
```

La clause `GROUP BY`

Ces options permettent de définir et de traiter des groupes. Un groupe est formé à partir d'un ensemble de lignes d'une table ayant une ou plusieurs caractéristiques communes.

Exemple : *Quel est le salaire moyen et le salaire minimum des employés à l'intérieur de chaque département pour les n° employés > 00010.*

```
SELECT WDEPT, AVG (SALAIRE), MIN (SALAIRE) FROM EMPLOYES
WHERE NOEMP > 00010
GROUP BY WDEPT
```

La clause `HAVING`

La clause `HAVING` est utilisée en conjonction avec la clause `GROUP BY` . La clause `HAVING` agit comme critère de sélection pour les groupes renvoyés avec la clause `GROUP BY` .

Exemple : *Quel est le salaire moyen et le salaire minimum des employés à l'intérieur de chaque département pour les n° employés > 00010 ? Lister uniquement les groupes pour lesquels la moyenne est supérieure a 16 000*

```
SELECT WDEPT, AVG (SALAIRE), MIN (SALAIRE) FROM EMPLOYES
WHERE NOEMP > 00010
GROUP BY WDEPT
HAVING AVG (SALAIRE) >= 16000
```

Il ne faut pas confondre les clauses `WHERE` et `HAVING` .

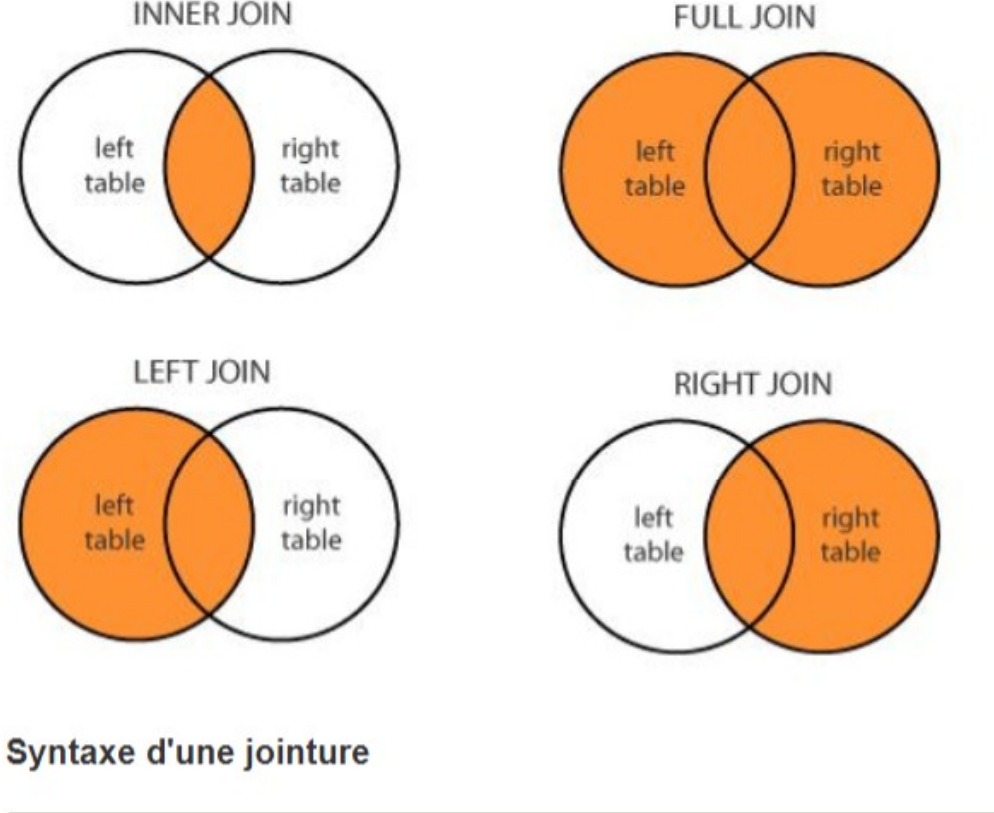
- `WHERE` permet de sélectionner des lignes avant la formation des groupes.
- `HAVING` permet de ne retenir que certains des groupes constitués par la clause `GROUP BY` .

L'opération de jointure - `JOIN`

- Une jointure sql `JOIN` combine l'enregistrement de deux tables.
- Une jointure `JOIN` localise les valeurs de colonne liées dans les deux tables.
- Une requête peut contenir zero, une, ou plusieurs operations de jointures.
- `INNER JOIN` = `JOIN` ; le mot clé `INNER` est facultatif.

Les differents types

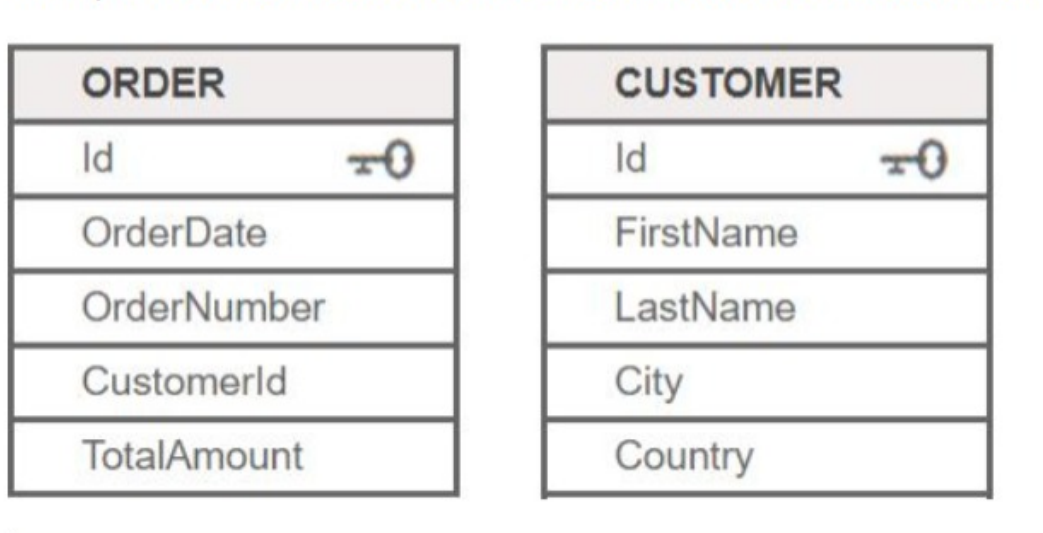
- `JOIN` : Sélectionne des enregistrements ayant des valeurs correspondantes dans l'ensemble des tables.
- `FULL JOIN` : Sélectionne tous les enregistrements correspondant aux enregistrements de gauche ou de droite.
- `LEFT JOIN` : Sélectionne l'enregistrement de la première table (la plus à gauche) avec les enregistrements correspondants de la droite.
- `RIGHT JOIN` : Sélectionne l'enregistrement de la deuxième table (la plus à droite) avec les enregistrements de table correspondants de gauche.



Syntaxe d'une jointure

```
SELECT *column-names*
  FROM *table_name1* JOIN *table_name2*
    ON *column_name1* = *column_name2*
 WHERE *condition*
```

Exemple : *Lister touters les commandes avec les informations clients*



ORDER
Id
OrderDate
OrderNumber
CustomerId
TotalAmount

CUSTOMER
Id
FirstName
LastName
City
Country

```
SELECT OrderNumber, TotalAmount, FirstName, LastName, City, Country
FROM [Order] JOIN Customer
ON [Order].CustomerId = Customer.Id
```

Resultat :

OrderNumber	TotalAmount	FirstName	LastName	City	Country
542378	440.00	Paul	Henriot	Reims	France
542379	1863.40	Karin	Josephs	Münster	Germany
542380	1813.00	Mario	Pontes	Rio de Janeiro	Brazil
542381	670.80	Mary	Saveley	Lyon	France
542382	3730.00	Pascale	Cartrain	Charleroi	Belgium
542383	1444.80	Mario	Pontes	Rio de Janeiro	Brazil
542384	625.20	Yang	Wang	Bern	Switzerland
...					