SQL
et
Bases de données

Cours 2

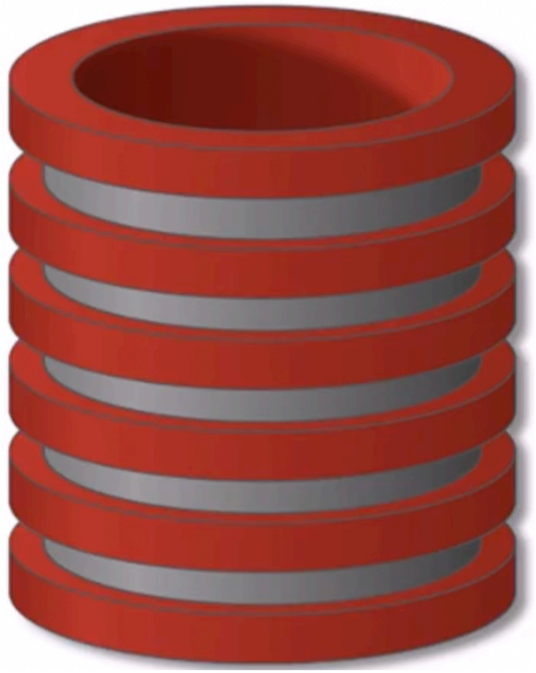
Jean-Jacques Lévy

jean-jacques.levy@inria.fr

http://jeanjacqueslevy.net/lp-sql

Plan

- installations de sqlite et VScode
- création d'une base de données
- requêtes élémentaires
- requêtes imbriquées
- modèle relationnel

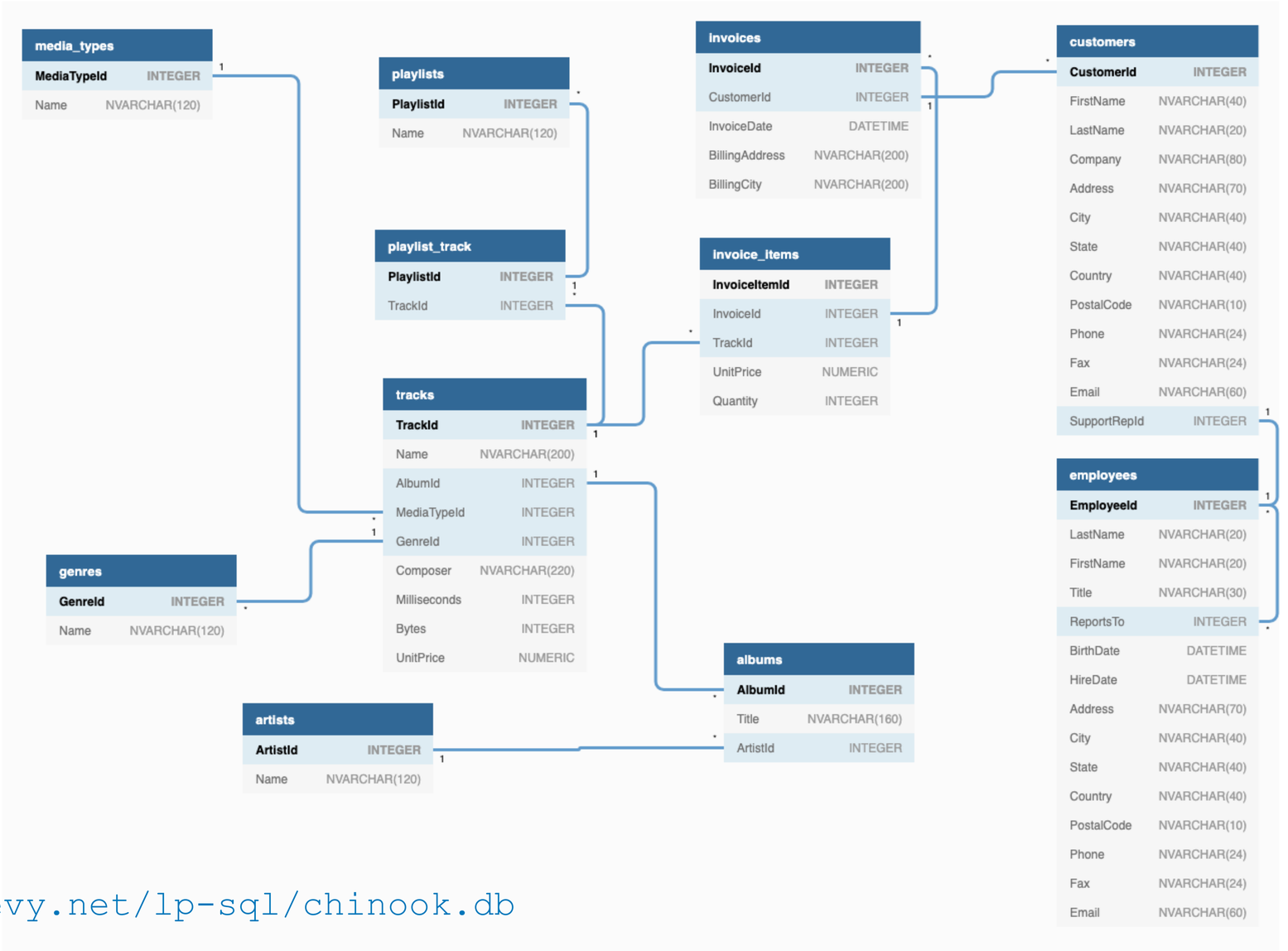


- un bon tutoriel

  http://www.w3schools.com/sql/default.asp

Exemple 2 du cours 1

- Chinook.db

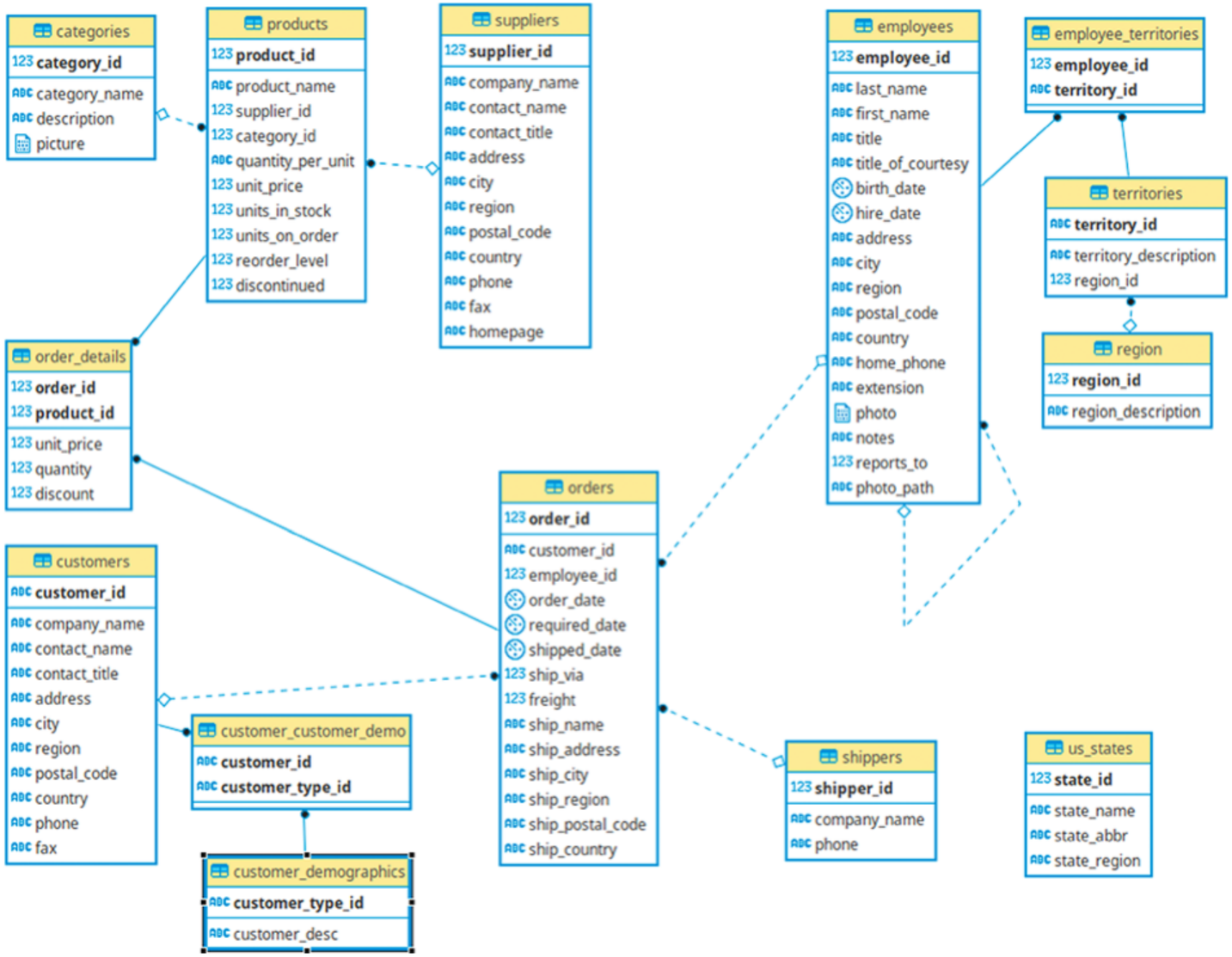


- on peut la charger en:

  http://jeanjacqueslevy.net/lp-sql/chinook.db

Exemple 3 du cours 1

- Northwind_small.sqlite



- on peut la charger en:

http://jeanjacqueslevy.net/lp-sql/northwind.db

Création d'une base de données en SQL

- définir les schémas

client

```
create table if not exists "client"
(
	[cID] integer primary key autoincrement NOT NULL,
	[nom] nvarchar(20),
	[actif] integer,
	[ville] nvarchar(10)
);
```

produit

```
create table if not exists "produit"
(
	[pID] integer primary key autoincrement NOT NULL,
	[pNom] nvarchar(10),
	[pCat] nvarchar(10),
	[pVille] nvarchar(10),
	[prix] integer
);
```

devis

```
create table if not exists "devis"
(
	[cID] integer NOT NULL,
	[pNom] nvarchar(10),
	[dCat] nvarchar(10),
	[commande] integer  -- 0 si devis en cours, pId du produit accepté
);
```

Initialisation des valeurs

- `autoincrement` met à jour automaitquement le numéro de série

```
INSERT INTO client (nom, actif, ville) VALUES ('Tom', 23000, 'Bordeaux');
INSERT INTO client (nom, actif, ville) VALUES ('Jean-Jacques', 38000, ‘Paris’);
…
```

- déclarer un champ `primary key` signifie que sa valeur est unique (utile pour la formation d’indexs)

```
INSERT INTO produit (pNom, pCat, pVille, prix) VALUES ('clio', 'auto', 'Paris', 13000);
INSERT INTO produit (pNom, pCat, pVille, prix) VALUES ('audi', 'auto', 'Paris', 45000);
…
```

```
INSERT INTO devis (cID, pNom, dCat, commande) VALUES (2, 'peugeot', 'velo', 0);
INSERT INTO devis (cID, pNom, dCat, commande) VALUES (7, NULL, 'velo', 0);
…
```

Quelques requêtes pour s’échauffer

• pour avoir la liste des tables en SQLite

```
.tables
```

• lister chaque table

```
select * from clients;
select * from produits;
select * from devis;
```

• lister les noms des clients qui figurent dans la table des devis

• lister les noms des clients avec leurs actifs en euros et en RMBs

• idem mais pour ceux qui ont plus de 30000 RMBs

• lister toutes les villes des clients et des produits

http://www.w3schools.com/sql/default.asp

Quelques requêtes pour s'échauffer

- lister les noms des clients qui figurent dans la table des devis (solution)

```
select nom,pNom,dCat
from devis,client
where client.cID = devis.cID;
```

- lister les noms des clients avec leurs actifs en euros et en RMBs
- idem mais pour ceux qui ont plus de 30000 RMBs

```
select nom, actif as actifEuro,
   actif*7.01 as actifRMB, ville
from client
where actifRMB > 300000;
```

- lister toutes les villes des clients et des produits

```
select distinct ville from client
union
select distinct pVille from produit;
```

http://www.w3schools.com/sql/default.asp

Quelques requêtes pour s’échauffer - bis

- lister dans l’ordre croissant ou décroissant

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

- valeur NULL de certainese données

```
SELECT column_names
FROM table_name
WHERE column_name IS NULL;
```

Requêtes imbriquées

- `as` pour renommer des variables (*alias*)

```
SELECT column_name AS alias_name
FROM table_name;
```

- lister le client avec le plus grand actif (sans utiliser `max`)

```
select * from client C1
where not exists (
  select * from client C2
  where C1.actif < C2.actif);
```

- lister le client avec le plus grand actif (avec `max`)

```
select max(actif), nom from client;
```

- lister le produit, la ville, l'actif et le nom du client avec le plus grand actif pour acheter ce produit

Ajouts, modifications, suppressions

- insérer de nouveaux n-uplets

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

- on peut aussi insérer une nouvelle table

- modification

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

Quelques requêtes plus ésotériques

- filtrage

```
SELECT column1, column2, ...
FROM table_name
WHERE columnN LIKE pattern;
```

- test d'appartenance à un ensemble

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...);
```

- union de 2 tables

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

- intersection de 2 tables

```
SELECT column_name(s) FROM table1
INTERSECT
SELECT column_name(s) FROM table2;
```

Groupes

- groupe de n-uplets

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```

- condition dans un groupe (on ne peut utiliser `where`)

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

Exists, any, all, commentaires

- test si une requête a un résultat vide

```
SELECT column_name(s)
FROM table_name
WHERE EXISTS
(SELECT column_name FROM table_name WHERE condition);
```

- ANY et ALL ne marchent pas avec sqlite (ok avec Postgres)

- commentaires avec des tirets

```
create table if not exists "devis"
(
        [cID] integer NOT NULL,
        [pNom] nvarchar(10),
        [dCat] nvarchar(10),
        [commande] integer  -- 0 si devis en cours, pId du produit accepté
);
```

Prochain cours

- le modèle relationnel
- correspondance avec la logique du 1er ordre
- introduction au requêtes concurrentes