Big Data Infrastructures
Fall 2018

Lab 01 : SQL Review

Author : Thomas Schaller, Sylvain Julmy

Professor : Philippe Cudré-Mauroux
Assistant : Akansha Bhardwaj

Exercice A

We use the following to create a new database :

```sql
CREATE DATABASE homework_1
WITH
OWNER = postgres
ENCODING = 'UTF8'
-- LC_COLLATE = 'French_Switzerland.1252'
-- LC_CTYPE = 'French_Switzerland.1252'
TABLESPACE = pg_default
CONNECTION LIMIT = -1;
```

Exercice B

We create the tables using the following queries :

Paper

```sql
CREATE TABLE Paper (
  paperID  integer primary key,
  title    char(255),
  abstract text
);
```

Author

```sql
CREATE TABLE Author (
  authorID    integer primary key,
  name        char(255),
  email       char(255),
  affiliation char(255)
);
```

Conference

```sql
CREATE TABLE Conference (
  confID  integer primary key,
  name    char(255),
  ranking integer
);
```

Writes

```sql
CREATE TABLE Writes(
  authorID integer,
  paperID integer,
  PRIMARY KEY (authorID, paperID),
  CONSTRAINT fk_writes_author
  REIGN KEY (authorID)
  FERENCES Author(authorID)
  DELETE CASCADE,
  CONSTRAINT fk_writes_paper
  REIGN KEY (paperID)
  FERENCES Paper(paperID)
  DELETE CASCADE
);
```

Submits

```sql
CREATE TABLE Submits(
  paperID integer,
  confID integer,
  isAccepted boolean,
  date date,
  PRIMARY KEY (paperID, confID),
  CONSTRAINT fk_submits_conf
  FOREIGN KEY (confID)
  REFERENCES Conference(confID)
  ON DELETE CASCADE,
  CONSTRAINT fk_submits_paper
  FOREIGN KEY (paperID)
  REFERENCES Paper(paperID)
  ON DELETE CASCADE
);
```

Cites

```sql
CREATE TABLE Cites(
  paperIDfrom integer,
  paperIDto integer,
  PRIMARY KEY (paperIDfrom, paperIDto),
  CONSTRAINT fk_cites_paperfrom
  FOREIGN KEY (paperIDfrom)
  REFERENCES Paper(paperID)
  ON DELETE CASCADE,
  CONSTRAINT fk_cites_paperto
  FOREIGN KEY (paperIDto)
  REFERENCES Paper(paperID)
  ON DELETE CASCADE
);
```

Note : We put all of the foreign key to ”ON DELETE CASCADE”, because for example, for the writes relation, if we delete a paper or an author, we have to delete also the writes row corresponding to this author.

Exercice C

In order to populate the database, we have written a Node.js application using knex. The following listings shows how we are doing it.

```javascript
// populate the paper table
var faker = require('faker');

let createRecord = (knex, id) => {
  return knex('paper').insert({
    paperid: id,
    title: faker.lorem.words(),
    abstract: faker.lorem.sentences(),
  })
}

exports.seed = (knex, Promise) => {
  return knex('paper').del()
    .then(() => {
      let records = [];

      for (let i = 1; i < 40; i++) {
        records.push(createRecord(knex, i))
      }

      return Promise.all(records);
    });
};
```

```javascript
// populate the author table
var faker = require('faker');

const AFFILATION = ['University of Fribourg', 'University of Bern', 'University of Neuchatel', 'EPFL',
↪  'HEIA', 'ETH'];

let createRecord = (knex, id) => {
  return knex('author').insert({
    authorid: id,
    name: faker.name.firstName() + " " + faker.name.lastName(),
    email: faker.internet.email(),
    affiliation: AFFILATION[Math.floor(Math.random() * 6)],
  })
}

exports.seed = (knex, Promise) => {
  return knex('author').del()
    .then(() => {
      let records = [];

      for (let i = 1; i < 15; i++) {
        records.push(createRecord(knex, i))
      }

      return Promise.all(records);
    });
};
```

```javascript
// populate the conference table
var faker = require('faker');

let createRecord = (knex, id) => {
  return knex('conference').insert({
    confid: id,
    name: faker.lorem.word(),
    ranking: Math.floor(Math.random() * 11),
  })
}

exports.seed = (knex, Promise) => {
  return knex('conference').del()
    .then(() => {
      let records = [];

      for (let i = 1; i < 15; i++) {
        records.push(createRecord(knex, i))
      }

      return Promise.all(records);
    });
};
```

```javascript
var faker = require('faker');

const primarykey_pair = [];

let createRecord = (knex, id) => {
  const random = Math.floor(Math.random() * 14) + 1;
  primarykey_pair.push(random+","+id);
  return knex('writes').insert({
    authorid: random,
    paperid: id,
  })
}

let createRecordRandom = (knex, id) => {
  let authorid = Math.floor(Math.random() * 14) + 1;
  let paperid = Math.floor(Math.random() * 39) + 1;
  while (primarykey_pair.indexOf(authorid+","+paperid) > -1) {
    authorid = Math.floor(Math.random() * 14) + 1;
    paperid = Math.floor(Math.random() * 39) + 1;
  }
  primarykey_pair.push(authorid+","+paperid);
  return knex('writes').insert({
    authorid,
    paperid,
  })
}

exports.seed = (knex, Promise) => {
  return knex('writes').del()
    .then(() => {
      let records = [];

      for (let i = 1; i < 40; i++) {
        records.push(createRecord(knex, i))
      }
      for (let i = 1; i < 15; i++) {
        records.push(createRecordRandom(knex, i))
      }

      return Promise.all(records);
    });
};
```

```javascript
var faker = require('faker');

const primarykey_pair = [];

let createRecord = (knex, id) => {
  let paperid = Math.floor(Math.random() * 39) + 1;
  let confid = Math.floor(Math.random() * 14) + 1;
  while (primarykey_pair.indexOf(paperid+","+confid) > -1) {
    paperid = Math.floor(Math.random() * 39) + 1;
    confid = Math.floor(Math.random() * 14) + 1;
  }
  primarykey_pair.push(paperid+","+confid);
  return knex('submits').insert({
    paperid,
    confid,
    isaccepted: faker.random.boolean(),
    date: faker.date.future(),
  })
}

exports.seed = (knex, Promise) => {
  return knex('submits').del()
    .then(() => {
      let records = [];

      for (let i = 1; i < 60; i++) {
        records.push(createRecord(knex, i))
      }

      return Promise.all(records);
    });
};
```

```javascript
var faker = require('faker');

const primarykey_pair = [];

let createRecord = (knex, id) => {
  let paperidfrom = Math.floor(Math.random() * 39) + 1;
  let paperidto = Math.floor(Math.random() * 39) + 1;
  while (primarykey_pair.indexOf(paperidfrom+","+paperidto) > -1) {
    paperidfrom = Math.floor(Math.random() * 39) + 1;
    paperidto = Math.floor(Math.random() * 39) + 1;
  }
  primarykey_pair.push(paperidfrom+","+paperidto);
  return knex('cites').insert({
    paperidfrom,
    paperidto,
  })
}

exports.seed = (knex, Promise) => {
  return knex('cites').del()
    .then(() => {
      let records = [];

      for (let i = 1; i < 40; i++) {
        records.push(createRecord(knex, i))
      }

      return Promise.all(records);
    });
};
```

Exercice D

1)

```sql
select affiliation, count(*)
from author
group by affiliation;
```

2)

```sql
select p.abstract, a.authorId
from paper as p
inner join writes as w
  on p.paperId = w.paperId
inner join author as a
  on w.authorId = a.authorId
where a.authorId = 2;
```

3)

```sql
create view PublishesIn1(authorID, confID) as
  select a.authorID, c.confID
    from author as a
    inner join writes as w
      on a.authorId = w.authorId
    inner join paper as p
      on w.paperId = p.paperId
    inner join submits as s
      on p.paperId = s.paperId
    inner join conference as c
      on s.confId = c.confId
    where s.isAccepted = true;
```

4)

```sql
select distinct(w.authorId)
from writes as w
inner join cites as c
  on c.paperIdFrom = w.paperId
inner join writes as w2
  on c.paperIdTo = w.paperId
where w.authorId = w2.authorId;
```

5)

```sql
select title
from paper
where paperid in (
  select paperId
  from writes
  where paperId in (
    select paperid
    from writes
    where authorid = 2
  )
  group by paperId
  having count(paperId) > 1
);
```