データベース工学
最終レポート

48-146427 谷川祐一

2014/08/18

1　概要

私は課題 1 のデータベースシステムの設計と性能評価に取り組んだ．以下にレポートを行う．

2　E-R 図

データベースシステムの E-R 図を図 1 に示す．営業拠点, 顧客, 部品, 注文はそれぞれ図 1 中では branches, customers，parts，orders と表記している．以下の 3 つの関係が存在する．

- branches：orders = 1：N
- customers：orders = 1：N
- parts：orders = 1：N

3　スキーマ設計

E-R 図を元に，branches，customers，parts，orders からなるスキーマを設計した．以下に各実体の属性の意味を述べ，クエリ 1 に SQL で示す．

3.1　各実体の属性の意味

3.1.1　branches (**営業拠点**)

b_id　重複しない拠点番号
b_name　営業所名
b_email　E-mail address
b_zipcode　郵便番号

3.1.2　customers (**顧客**)

c_id　重複しない顧客番号
c_last_name　姓
c_first_name　名

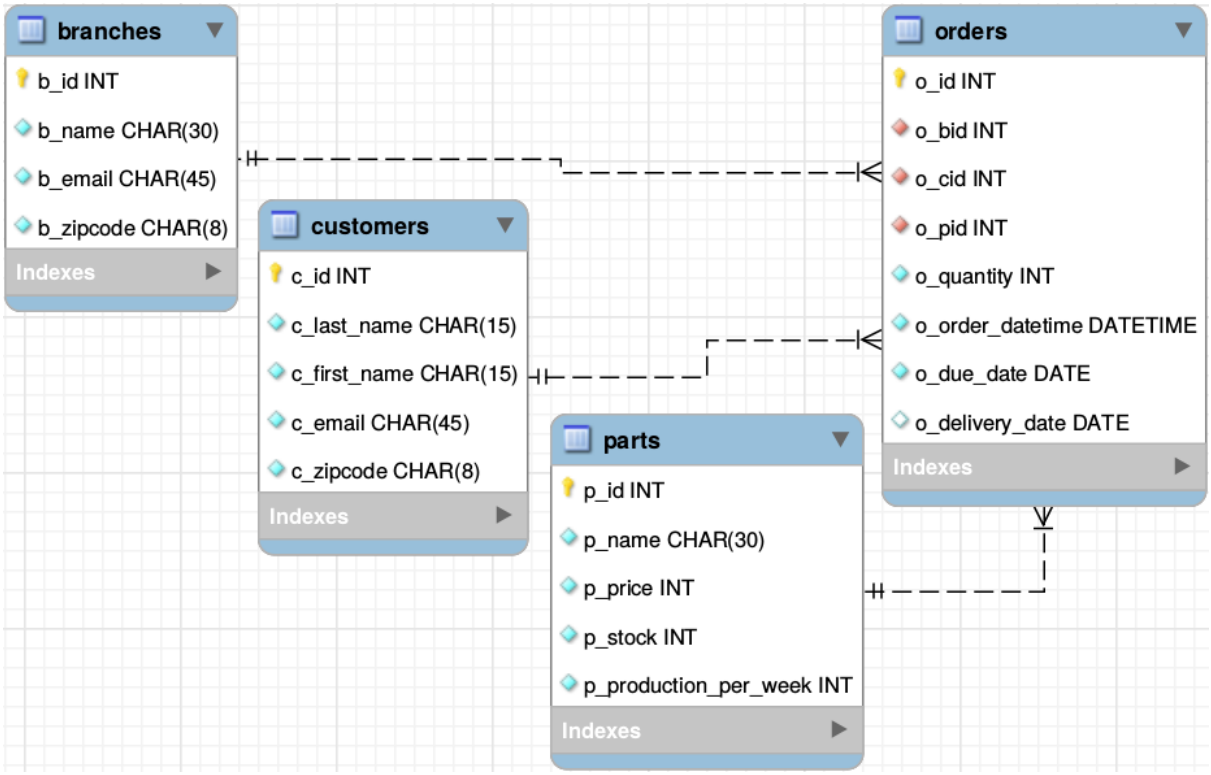

図 1　E-R 図

c_email　E-mail address
c_zipcode　郵便番号

3.1.3　parts (**部品**)

p_id　重複しない部品番号
p_name　部品名
p_price　価格
p_stock　在庫量
p_production_per_week　1 週間に製造できる量

3.1.4　orders (**注文**)

o_id　重複しない注文番号
o_bid　注文を受けた営業拠点の拠点番号
o_cid　注文を出した顧客の顧客番号
o_pid　注文された部品の部品番号
o_quantity　注文した個数
o_order_datetime　注文日時
o_due_date　見込み納期日

o_delivery_date　発送日

3.2　SQL

クエリ 1　スキーマのクエリ

```
CREATE TABLE `branches` (
  `b_id` INT NOT NULL,
  `b_name` CHAR(30) NOT NULL,
  `b_email` CHAR(45) NOT NULL,
  `b_zipcode` CHAR(8) NOT NULL,
  PRIMARY KEY (`b_id`));

CREATE TABLE `customers` (
  `c_id` INT NOT NULL,
  `c_last_name` CHAR(15) NOT NULL,
  `c_first_name` CHAR(15) NOT NULL,
  `c_email` CHAR(45) NOT NULL,
  `c_zipcode` CHAR(8) NOT NULL,
  PRIMARY KEY (`c_id`));

CREATE TABLE `parts` (
  `p_id` INT NOT NULL,
  `p_name` CHAR(30) NOT NULL,
  `p_price` INT NOT NULL,
  `p_stock` INT NOT NULL,
  `p_production_per_week` INT NOT NULL,
  PRIMARY KEY (`p_id`));

CREATE TABLE `orders` (
  `o_id` INT NOT NULL,
  `o_bid` INT NOT NULL,
  `o_cid` INT NOT NULL,
  `o_pid` INT NOT NULL,
  `o_quantity` INT NOT NULL,
  `o_order_datetime` DATETIME NOT NULL,
  `o_due_date` DATE NOT NULL,
  `o_delivery_date` DATE NULL,
  PRIMARY KEY (`o_id`),
  FOREIGN KEY (`o_bid`) REFERENCES `branches` (`b_id`),
  FOREIGN KEY (`o_cid`) REFERENCES `customers` (`c_id`),
  FOREIGN KEY (`o_pid`) REFERENCES `parts` (`p_id`));
```

4　問い合わせ SQL クエリ

以下の問い合わせ A,B,C を行う SQL クエリをそれぞれ示す.

4.1　A. 1 個 3,000 円以上の部品を 1 度でも買ったことのある顧客の姓名を出力

クエリ 2 と 3 の 2 通り考えられる.

クエリ 2　問い合わせ A のクエリ (1)

```
SELECT DISTINCT
```

```
    c_last_name, c_first_name
FROM
    customers, parts, orders
WHERE
    c_id = o_cid
        AND p_id = o_pid
    AND p_price >= 3000;
```

クエリ 3　問い合わせ A のクエリ (2)

```
SELECT
    c_last_name, c_first_name
FROM
    customers
WHERE
    EXISTS (
        SELECT
            *
        FROM
            orders, parts
        WHERE
            c_id = o_cid
            AND p_id = o_pid
            AND p_price >= 3000);
```

4.2　B. 自分の住む同じ都道府県にある営業拠点からだけ購入している顧客の姓名を出力

クエリ 4　問い合わせ B のクエリ

```
SELECT
    c_last_name, c_first_name
FROM
    customers
WHERE
    SUBSTRING(c_zipcode, 1, 2) = ALL (
        SELECT
            SUBSTRING(b_zipcode, 1, 2)
        FROM
            branches,
            orders
        WHERE
            c_id = o_cid
            AND b_id = o_bid);
```

4.3　C. 全ての注文についての平均の納品日数を出力

クエリ 5　問い合わせ C のクエリ

```
SELECT
    AVG(DATEDIFF(o_delivery_date, o_order_datetime))
FROM
    orders;
```

5 SQL クエリの性能測定

前述の SQL クエリについて，RDS を利用して性能測定を行った．測定環境，利用したデータセット，測定方法，測定結果を順に述べる．

5.1 測定環境

DB Engine mysql
DB Engine Version 5.6.17
DB Instance Class db.t1.micro, 1 vCPU, 0.613 GiB RAM
Multi-AZ Deployment No
Allocated Storage 5 GB
Use Provisioned IOPS No
innodb_buffer_pool_size 295 MiB

5.2 利用したデータセット

TPC-H ベンチマーク [1] のスキーマを参考にしてテストデータを生成するプログラムを作成し，それを利用してテストデータを生成した．以下に生成アルゴリズムと生成したデータサイズについて述べる．

5.2.1 生成アルゴリズム

以下に各属性の生成アルゴリズムを述べる．問い合わせ A と B において 0 や 1 のような極端なセレクティビティを避けてある程度のセレクティビティを確保するための生成規則については，5.2.3 と 5.2.4 で後述する．

アルゴリズム中では名前や E-mail address などの値は現実には意味をなさないものとなっているが，テストデータとしては妥当だと考える．

branches

b_id INT 1 から始まる一意な連番の整数.

b_name CHAR(30) ”Branch♯000000001”のように，”Branch♯”の後に b_id を 9 桁の右詰めゼロ詰めの固定長の文字列にして結合した文字列.

b_email CHAR(45) ”Branch♯000000001@example.com”のように，b_name の後に”@example.com”を結合した文字列.

b_zipcode CHAR(8) ”ddd-dddd”なる文字列．d は 0 から 9 のランダムな数値とする．ただし後述する 5.2.4 の生成規則を適用する．

customers

c_id INT 1 から始まる一意な連番の整数.

c_last_name CHAR(15) ”Customer”なる文字列.

c_first_name CHAR(15) ”♯000000001”のように，”♯”の後に c_id を 9 桁の右詰めゼロ詰めの固定長の文字列

にして結合した文字列.

c_email CHAR(45)　”Customer♯000000001@example.com”のように，c_last_name, c_first_name, ”@example.com”を結合した文字列.

c_zipcode CHAR(8)　”ddd-dddd”なる文字列．d は 0 から 9 のランダムな数値とする．ただし後述する 5.2.4 の生成規則を適用する.

parts

p_id INT　1 から始まる一意な連番の整数.

p_name CHAR(30)　”Part♯000000001”のように，”Part♯”の後に p_id を 9 桁の右詰めゼロ詰めの固定長の文字列にして結合した文字列.

p_price INT　100 から 3050 までのランダムな整数．範囲の決め方は後述する 5.2.3 による.

p_stock INT　100 から 1000 までのランダムな整数．性能測定するクエリのセレクティビティには関係ないので，範囲は適当に決めた.

p_production_per_week INT　100 から 1000 までのランダムな整数．同上.

orders

o_id INT　1 から始まる一意な連番の整数

o_bid INT　1 から b_id の最大値までのランダムな整数．ただし後述する 5.2.4 の生成規則を適用する.

o_cid INT　1 から c_id の最大値までのランダムな整数．ただし後述する 5.2.4 の生成規則を適用する.

o_pid INT　1 から p_id の最大値までのランダムな整数.

o_quantity INT　1 から 50 までのランダムな整数.

o_order_datetime DATETIME　2013 年 0 時 0 分 0 秒から 2013 年 12 月 16 日 23 時 59 分 59 秒までのランダムな日時.

o_due_date DATE　納期までの日数 due を 1 から 10 のランダムな整数とし，due を o_order_datetime の日付に加えた日付.

o_delivery_date DATE　due に 0.5 から 1.5 までのランダムな実数 (プログラム上は double) を掛けたものを o_order_datetime の日付に加えた日付．o_delivery_date が最大でも 2013 年 12 月 31 日に収まるようにこれら 3 つの日時と日付を設定した．また問い合わせ C の平均納品日数は約 5.5 日となる.

5.2.2 データサイズ

表 1 に示すサイズのデータを作成した．容量の概算には表 2 に示すデータ型のバイト数 [2] を用いた.

5.2.3 問い合わせ A のセレクティビティのための生成規則

問い合わせ A の customers に対するセレクティビティの概算は，3,000 円以上の部品の割合を p，顧客 1 人当たり平均注文回数を n とすると，

$$1 - (1 - p)^n$$

となる．生成したデータサイズから $n = 20$ であり，部品の価格のランダムな 100 から 3,050 の整数とすると $p = 0.017$ となり，セレクティビティは 0.29 となる.

表 1　データサイズ

テーブル	行数	容量 (概算) [MB]
branches	10,000	0.82
customers	150,000	13.05
parts	200,000	9.2
orders	3,000,000	93
合計	-	116.07

表 2　データ型のバイト数

INT	CHAR(X)	DATETIME	DATE
4	X	5	3

5.2.4　問い合わせ B のセレクティビティのための生成規則

問い合わせ B の customers に対するセレクティビティの概算は，顧客が同じ都道府県の営業拠点から注文する確率を q，顧客 1 人当たり平均購入回数を n とすると，

$$q^n$$

となる．生成したデータサイズから $n = 20$ であり，仮に $q = 0.9$ とすると，セレクティビティは 0.1 となる．

$q = 0.9$ を実現するために，以下のような生成規則を導入する．

b_zipcode と c_zipcode　それぞれ先頭 2 文字を，b_id の下 2 桁，c_id の下 2 桁とする．すなわち拠点番号と顧客番号の下 2 桁で都道府県を表すようにする．

o_bid と o_cid　0.9 の確率で下 2 桁を同じ番号として生成する．

5.3　測定方法

問い合わせ A のクエリ 2 と 3，問い合わせ B のクエリ 4，問い合わせ C のクエリ 5 について，それぞれ外部キーのインデックスの有無の 2 通りについて 5 回づつ実行して，実行時間の平均と標準偏差を求めた．またクエリ 2 については，外部キーのインデックスに加えて p_price にセカンダリインデックスを作成した場合についても測定を行った．

なおキャッシュの効果を回避するために以下の方法で測定を行った．まず，テストデータとは別に 800MB ほどのダミーデータをデータベースに作成した．各問い合わせクエリの実行前には，ダミーデータのテーブルをクエリ 6 を用いてフルテーブルスキャンを行い．その後 RDS の DB instance を再起動させた．

RDS では DBMS より低レイヤのキャッシュを明示的にクリアする方法は見当たらなかったので，0.613 GiB のメモリサイズを超えるダミーデータを読み込むことで実質的にキャッシュからテストデータを追い出したとみなした．DBMS のキャッシュについては，DB instance を再起動することでクリアしたとみなした．

クエリ 6　ダミーデータをフルスキャンする

```
1 SELECT
2     count(*)
```

```
3 FROM
4     dummy;
```

5.4 測定結果

実行時間の平均と標準偏差を表 3 に示す.

外部キーインデックスがない場合のクエリ 3 とクエリ 4 はそれぞれ 2〜3 時間ほど実行してみたものの終わらなかったので，今回は測定不能とみなした.

表 3　問い合わせクエリの実行時間の平均と標準偏差

問い合わせ	クエリ	インデックス	平均 [秒]	標準偏差 [秒]
A	2	外部キー	22.92	0.15
	2	外部キー + p_price	21.33	0.22
	2	-	18.38	0.16
	3	外部キー	31.16	0.65
	3	-	測定不能	測定不能
B	4	外部キー	25.29	0.86
	4	-	測定不能	測定不能
C	5	外部キー	13.44	0.09
	5	-	13.57	0.3

5.5 考察

問い合わせ A のクエリ 2 で外部キーインデックスがある場合の実行計画は，parts テーブルにアクセスした後に外部キーを用いて orders テーブルをジョインし，その後プライマリキーを用いて customers テーブルをジョインするというものだった．p_price のセカンダリインデックスの有無で少しだけしか実行時間が変わらなかったのは，p_price の値をインデックスから読むか実テーブルから読むかにかかる時間よりも，他のテーブルとのジョインするためにディスクに対するランダムアクセスをする時間が支配的だからだと考えられる．今回は実験できなかったが，3,000 円以上の部品の割合が今よりも極端に 0 に近い場合だと，p_price のセカンダリインデックスは有効に働くと考えられる.

また外部キーインデックスがない場合の 2 の実行計画は，orders テーブルにアクセスした後に customers テーブルと parts テーブルをそれらのプライマリキーを用いてジョインするというものだった．このとき実行時間が外部キーありの場合よりも短くなったのは，MySQL のデータ構造上セカンダリインデックスを使う場合，セカンダリキーのインデックスを引いて取得したプライマリキーの値で実データを取得するということを行っているので，セカンダリインデックスを引くという二度手間をせず直接プライマリキーでデータにアクセスできたためだと考えられる．MySQL のオプティマイザが本当に最適な実行計画を出せるかいうと，そうではないということが窺い知れる.

問い合わせ A のクエリ 3 と問い合わせ B のクエリ 4 の実行計画は，customers テーブルにアクセスした後，キーを用いずに相関サブクエリが実行されていた．サブクエリ中では，orders テーブルにアクセスした後，プライマリキーを用いてそれぞれ parts と branches テーブルをジョインしていた．キーを用いない相関サブク

エリなのでネスティッドループが用いられていると仮定すると，その時間計算量はテーブルの行数に換算して

$$(\text{外部クエリの行数}) \cdot (\text{内部クエリの行数}) = |\text{customers}| \cdot |\text{orders}| = 15,000 \cdot 3,000,000 = 4.5 \cdot 10^{11}$$

と大きいことがわかる．

問い合わせ C のクエリ 5 については orders テーブルをフルテーブルスキャンするので，外部キーインデックスの有無で実行時間は変化しない．

参考文献

[1] TPC Benchmark^{TM} H Standard Specification Revision 2.17.0 (Apr. 2014).

[2] MySQL :: MySQL 5.6 Reference Manual :: 11.7 Data Type Storage Requirements. http://dev.mysql.com/doc/refman/5.6/en/storage-requirements.html