1. MISC : notes de TP

- Everytime we change something in the source code :
    - Reinstall postgresql from source
    - make, sudo make install, start again, etc. see details
- Print : attention à bien flush

2. Traducteur chinois (PGSQL) -> Français

2.1. Histogrammes calculés

Tout d'abord, rendez-vous dans `rangetypes_typanalyze.c`. C'est là où se calculent les stats des colonnes de type `range`.

Dans cette fonction, on voit deux choses calculées :

1. *Bound-histogram* : un histogramme des bornes. En fait, on store dedans, comme on peut le voir à la ligne qui fait

    ```
    bound_hist_values[i] = PointerGetDatum(range_serialize(typcache,
                                                           &lowers[pos],
                                                           &uppers[pos],
                                                           false));
    ```

    un pointeur (Datum est un pointeur) vers un objet `range` sérialisé à partir des lower et upper bounds.

    En fait, en se documentant un peu, cet histogramme est un histogramme de *range* et contient une fusion des histogrammes lower-bound et upper-bound (cf. ce qui est dit dans le TP4). Donc cet histogramme de `range` contient l'information de 2 histogrammes de scalaires. On verra par la suite que pour calculer la selectivity, on va récupérer cet histogramme et le désérialiser pour en effet construire les lower-bound et upper-bound histogrammes.

2. *Length-histogram* : le nom est beau mais je n'ai aucune idée de ce qu'il représente. Cependant, avec un peu de documentation, on lit la chose suivante :

    A "length histogram" slot describes the distribution of range lengths in rows of a range-type column. stanumbers contains a single entry, the fraction of empty ranges. stavalues is a histogram of non-empty lengths, in a format similar to STATISTIC_KIND_HISTOGRAM: it contains M (>=2) range values that divide the column data values into M-1 bins of approximately equal population. The lengths are stored as float8s, as measured by the range type's subdiff function. Only non-null rows are considered.

    Source : `pg_statistics_d.h`, pour `#define STATISTIC_KIND_RANGE_LENGTH_HISTOGRAM`

2.2. Calculs de selectivity

2.2.1. Traduction du Chinois mandarin (BOUNDS HISTOGRAM) au Chinois traditionnel (LOWER-BOUND, `UPPER-BOUND HISTOGRAM`S)

On suit le TP4, particulièrement pour l'exo 7. On regarde la selectivity de l'opérateur *Overlaps*. Son OID est `OID_RANGE_OVERLAP_OP`, donc on fait un CTRL-F sur sa gueule dans `rangetypes_selfuncs.c` : on tombe dans la fonction `calc_hist_selectivity` qui va faire la selectivity pour les range type (logique).

On suit encore les explications du TP et on lit le code : le calcul de cette selectivity est la même que pour des scalaires en fait. Cependant c'est là qu'on va récupérer l'histogramme qui a été calculé dans `rangetypes_typanalze.c`.

Tout commence à la ligne 397, où on récupère l'histogramme de type `BOUNDS HISTOGRAM` comme indiqué dans l'argument de la fct :

```
    get_attstatsslot(&hslot, vardata->statsTuple,
                     STATISTIC_KIND_BOUNDS_HISTOGRAM, InvalidOid,
                     ATTSTATSSLOT_VALUES)))
```

et on le place dans l'objet `&hslot`, objet d'intérêt particulier.

En effet, quelques lignes plus tard, après avoir fait de la place pour des upper-bound et lower-bound histogrammes, on va :

- récupérer les range sérialisées qui se trouvent dans le *bounds histogram*,
- Désérialiser ces gentilles dames
- Placer le résultat de cette désérialisation dans les lower- et upper-bound histogrammes qu'on vient de créer. Tout ceci est dans les environs de la ligne 419 :

  ```
  range_deserialize(typcache, DatumGetRangeTypeP(hslot.values[i]),
                    &hist_lower[i], &hist_upper[i], &empty);
  ```

2.2.2. Traduction du Chinois traditionnel au Français

Maintenant qu'on a les histogrammes scalaires `hist_lower` et `hist_upper`, on peut calculer les selectivity par la fonction `calc_hist_selectivity_scalar`.

Dans cette fonction, on trouve le calcul de la selectivity.

```
index = rbound_bsearch(typcache, constbound, hist, hist_nvalues, equal);
selec = (Selectivity) (Max(index, 0)) / (Selectivity) (hist_nvalues - 1);
```

3. Arbre généalogique

On pourrait croire que ces fonctions sont consanguines mais en fait non ça va, du moins si on accepte les ébats entre cousins. En gros il y a du sens dans tout ce qu'on fait.

Voilà l'ordre des choses, qu'on obtient avec des CTRL-F successifs :

1. `pg_operator.dat` renseigne quelle fonction est appelée par quel opérateur. On voit que pour `OVERLAPS`, c'est `rangesel` qui est appelée. Elle se trouve dans `rangetypes_selfuncs.c`
2. `rangesel` appelle une fonction `calc_rangesel`
3. `calc_rangesel` appelle `calc_hist_selectivity`
    - Cette fonction va désérialiser les histogrammes calculés dans `rangetypes_typanalze.c`
    - et appeler `calc_hist_selectivity_scalar`
4. `calc_hist_selectivity_scalar` calcule une selectivity de manière classique

Sinon, dans une approche bottom-up, pour écrire une nouvelle structure, on a :

1. On calcule un histogramme et on l'encode dans le même objet où les bounds et length histogrammes sont encodés, à savoir, dans l'array `stats->stavalues`. Ici, par exemple, le BOUNDS HISTOGRAM est à un certain slot, il suffit de lire le code pour voir qu'on le stock dans `stats->stavalues[slot_idx]`.
2. Il faut définir un code similairement à `STATISTIC_KIND_BOUNDS_HISTOGRAM`, **mais il faut voir comment faire le lien entre ce code et le slot auquel correspond notre structure** !!
3. On le stocke dans `calc_rangesel` avec `get_attstatsslot`
4. Dans la fonction qui calcule la selectivity pour l'opérateur qu'on veut, on désérialise si nécessaire notre structure de données, et on l'utilise à bon escient

---

Putain bonne nuit