Quelques rappels sur ce que l'on a fait

« Quoi, l'interro c'est déjà samedi ?? »

Algèbre et calcul relationnels :

Le but du jeu est de représenter les données qui nous intéressent de manière un peu plus mathématique. L'algèbre relationnelle nous permet de les exprimer en termes d'opérations à effectuer sur un ensemble de relations. Dans le calcul relationnel, on s'intéresse plus à une description logique des choses, i.e. on s'attache aux conditions logiques que doivent satisfaire les éléments de la relation qui nous intéressent.

Dans les deux cas, les relations (= tables SQL) sont des ensembles au sens de la théorie des ensembles. Cette contrainte nous impose notamment de manipuler des éléments non comparables entre eux et tous différents les uns des autres. Le principe s'applique également aux nouvelles relations créées par restriction de nos relations initiales. Ainsi, certaines requêtes SQL ne peuvent pas être reproduites en algèbre ou en calcul relationnels (« sélectionner en ordonnant sur tel ou tel champs », etc.).

Plus précisément, une table X composée d'un entier et d'une chaîne de caractères sera représentée par un ensemble X inclus dans l'ensemble E de tous les couples (entier, chaîne) possibles. En particulier, X contient toutes les données « cohérentes », qui apparaissent dans notre table, alors que l'on peut trouver n'importe quoi dans E-X. C'est notamment à cause de cette dernière contrainte primordiale que le prédicat d'appartenance à X revient à chaque fois dans le calcul relationnel.

Algèbre relationnelle :

Principaux opérateurs :

- σ <condition_1>, …, <condition_k> (<relation>) : équivalent du WHERE en SQL.
- π <champs_1>, …, <champs_k> (<relation>) : équivalent du SELECT en SQL.
- <relation_1> × <relation_2> : produit cartésien de deux relations.
- <relation_1> ⋈ <relation_2> : jointure naturelle de deux relations.

Les deux premières opérations prennent une relation (<relation>) en paramètre et renvoient une nouvelle relation affinée selon les critères spécifiés. Les deux dernières prennent deux relations en paramètres (<relation_1> et <relation_2>) et renvoient une nouvelle relation « composée ». Ces deux opérations étant associatives et commutatives, il est possible d'en enchaîner plus de deux sans s'embêter avec le parenthésage.

Calcul relationnel :

Du point de vue de la terminologie, tout ensemble est encadré par des accolades qui expliquent d'où proviennent ses éléments : d'abord on spécifie la nature des éléments de l'ensemble (singletons, couples, n-uplets, etc.), puis on détaille comment ils ont été obtenus i.e. quelles conditions logiques sur tels ou tels ensembles nous ont permis de les obtenir.

{(<champs_1>, …, <champs_n>) / <condition_1> ∧ … ∧ <condition_k>}

Les parenthèses sont importantes : si elles sont absentes, alors plutôt que d'obtenir un ensemble de n-uplets vous aurez un ensemble global de singletons dans lequel toutes les valeurs possibles nageront de manière anarchique. En plus de ce problème, vous risquez de probablement perdre de l'information puisque les ensembles ne contiennent pas de doublons (si deux personnes ont 20 ans, cette information est redondante d'un point de vue général, mais ce n'est pas le cas si elle est accompagnée d'un identifiant unique).

Au niveau des conditions, la première chose à faire est de s'assurer que les attributs des éléments de l'ensemble viennent bien de la relation que l'on manipule. Par exemple, si vous voulez tous les éléments (entier, chaîne) qui sont cohérents (i.e. ceux de l'ensemble X), il faut s'assurer de ne pas prendre ceux de E-X :

{(entier, chaîne) / (entier, chaîne) ∈ X}

Si vous ne voulez sélectionner que les entiers de la relation X, notez que l'on a besoin des chaînes. En l'occurrence, on s'intéresse aux entiers qui apparaissent dans des éléments de X et tels qu'il existe une chaîne qui forme un élément de X avec cet entier.

{(entier) / ∃ chaîne : (entier, chaîne) ∈ X}

La méthode est TOUJOURS LA MÊME : d'abord on « sélectionne » les champs qui nous intéressent, puis on introduit ceux qui nous manquent pour respecter la logique de nos données et introduire nos contraintes.

Opérations ensemblistes :

Les opérations classiques de la théorie des ensembles sont bien entendu utilisables en algèbre et calcul relationnels puisque l'on manipule des ensembles. En particulier :

- L'union de deux ensembles (A∪B) : éléments appartenant à A ou B (de manière large).
- L'intersection de deux ensembles (A∩B) : éléments appartenant à la fois à A et B.
- La différence de deux ensembles (A-B) : éléments de A n'appartenant pas à B.

Ces opérations s'effectuent sur deux ensembles de même nature ! Vous ne pouvez pas faire l'union d'un ensemble de 2-uplets et d'un ensemble de 3-uplets. C'est également le cas en SQL : les champs des deux requêtes dont vous faîtes l'union doivent correspondre, au moins en ce qui concerne leurs types.

Notation en algèbre relationnelle : <relation_1> ∪/∩/- <relation_2>
Notation en calcul relationnel : {…} ∪/∩/- {…}
Notation en SQL : *Select*… *From*… *Union*/*Intersect*/*Except Select*… *From*…

Tout ce qu'on a vu, en vrac :

- Le produit cartésien et les jointures naturelles, externes gauches/droites/totales + comment les réaliser en algèbre et calcul relationnels.
- Trier des résultats par nom ou numéro de champs.
- Affiner des résultats via le *Where* : *is null*, *is not null*, *exists*, *unique*.
- Le regroupement avec *rollup*, *cube* et *grouping sets* + affiner les groupes selon certains critères (*Having*).
- Les différentes fonctions d'agrégation.
- Les requêtes imbriquées grâce aux mots-clefs *in*, *not in*.
- Diverses requêtes récentes : *rank over* (Q32), *partition* (Q33), *case when then* (Q34), *extract from* (Q35), *nullif* + *coalesce* (Q36), *with* (Q37), *match* (Q38).