12. Historique des données

Dans ce court chapitre, qui nous fournit également l'occasion de traiter de nouveaux exemples, nous montrons comment représenter dans une base de données l'historique des valeurs d'un objet.

Sautons immédiatement dans le vif du sujet. Nous recevons la tâche d'enregistrer dans la base de données l'historique des salaires perçus et des postes de travail occupés par les employés dans l'entreprise.

Des sauvegardes de la base de données sont évidemment effectuées sur une base régulière. En principe elles contiennent, à condition de les conserver suffisamment longtemps, toutes les valeurs passées. Mais cela ne nous permet pas de produire facilement et rapidement un historique du parcours de chaque employé.

Postes occupés et salaires perçus ne sont évidemment pas les seuls éléments de données qui changent en fonction du temps. Les prix des articles en vente, les cours de change pratiqués, les taux de TVA appliqués et bien d'autres grandeurs évoluent aussi. Souvent l'on est obligé de revenir aux valeurs du passé pour présenter l'évolution des affaires, effectuer des révisions ou corriger des transactions erronées. Et souvent cela pose problème parce que les anciennes valeurs ne sont plus dans le système, ayant simplement été recouvertes par les nouvelles.

Revenons à notre exemple.

Une solution scolaire résoudrait le problème employé – poste – salaire de la façon suivante:

Poste (#-poste, description, …

Employé (#-employé, nom, prénom, date naissance, sexe, salaire, *#-poste*, …

Cette solution ne permet pas de gérer l'historique des postes et salaires. On peut déjà l'améliorer en ajoutant des dates pour le salaire et le poste dans *employé*, mais ce n'est qu'un début.

Employé (#-employé, nom, prénom, date naissance, sexe, salaire, date salaire, *#-poste*, date poste, …

Il faudrait à présent poser la question suivante aux personnes chargées des ressources humaines: changements de salaires et changements de postes sont ils corrélés?

Si la réponse est OUI ou EN GÉNÉRAL OUI nous proposons la solution suivante:

Historique salaire postes (*#-employé*, date, salaire, *#-poste*

À chaque changement de salaire ou de poste on créée un nouvel objet dans cette table.

Si la réponse est clairement NON, par exemple parce que les salaires changent surtout en fin d'années et que le poste occupé change rarement ou ne donne pas automatiquement droit à un changement de salaire, on peut dissocier les deux historiques:

Historique salaire (*#-employé*, date, salaire

Historique postes (*#-employé*, date, *#-poste*

[Le lecteur se rappellera peut-être que nous avons parlé au chapitre 6 de la 5ème forme normale qui interdit la présence de plusieurs dépendances multiples non-corrélées. S'il n'existe donc aucune corrélation entre l'historique des salaires et celui des postes, la première solution viole en principe la 5ème forme normale. Son unique conséquence néfaste est pourtant de répéter une valeur de salaire déjà connue en cas de changement de poste ou une valeur de poste déjà connue en cas de changement de salaire. Il n'en résulte par contre aucune restriction gênante pour la gestion.]

Une fois ces historiques créés (peu importe qu'il s'agisse de la 1ère ou de la 2ème solution), faut-il garder les attributs *#-poste* et *salaire* (qui reflètent les valeurs actuelles) dans *employé*? Ce sont bien sûr des valeurs redondantes, puisqu'également disponibles dans l'historique où il suffit pour les obtenir de sélectionner l'objet avec la date la plus récente. SQL nous permet d'ailleurs de créer une vue liant les tables *employé* et *historique* de manière à nous présenter directement les données de l'employé avec le salaire et le poste actuels.

Nous gardons finalement la solution suivante:

Employé (#-employé, nom, prénom, date naissance, sexe, …

Poste (#-poste, description, …

Historique salaire postes (*#-employé*, date, salaire, *#-poste*, …

Cette solution possède également l'avantage que les personnes qui travaillent avec le fichier des employés n'ont pas directement accès à leur salaire, question de sécurité qui ne peut pas être ignorée lors de la conception d'une base de données. Rappelons pourtant qu'il s'agit ici d'un exercice destiné à illustrer comment décrire un historique. Dans la réalité, les salaires des employés sont en général gérés dans des modules séparés et n'apparaissent pas dans la base de données de gestion d'une organisation.

Traitement systématique des historiques

Revenons au problème général de l'historique des données. D'autres données liées à un employé peuvent évidemment changer: son nom de famille en cas de mariage, son adresse, son statut marital, son droit à des allocations de famille ou enfants, etc.

En général des solutions telles que:

Employé (#-employé, nom, prénom, **nom de jeune fille**, adresse, **ancienne adresse**, …

Article (#-article, libellé, prix, **ancien prix**, …

quoique couramment utilisées, sont limitatives et sources de problèmes. Que faire lorsqu'on a besoin de l'avant-dernière valeur ou de valeurs encore plus anciennes?

La solution idéale serait de gérer systématiquement l'historique de tous les types d'objets dont certaines valeurs sont susceptibles de changer. Pour le faire, on crée un nouvel objet à chaque changement d'une valeur quelconque et on fait figurer dans la clé primaire la date et, puisque nous y sommes, l'heure de création de cet objet:

Employé (#-employé, date-heure, nom, prénom, adresse, …

Article (#-article, date-heure, libellé, prix, …

On dit que les objets portent un tampon horaire, timestamp en anglais.

Une telle approche entraîne évidemment la répétition de beaucoup d'informations dans la base de données (toutes les valeurs qui ne changent pas!), mais le volume nécessaire au stockage des données ne possède plus la même importance que par le passé (aujourd'hui c'est plutôt le foisonnement d'images et vidéos qui pose problème au niveau du stockage). Néanmoins, gérer systématiquement comme décrit ci-dessus l'historique de **tous** les types d'objet alourdirait considérablement le système et ne se justifie en général pas.

En contrepartie, cette approche décrit une base de données dans laquelle un objet **n'est jamais modifié** après avoir été enregistré, puisqu'une nouvelle version de l'objet est créée à chaque changement, même mineur. C'est une situation claire, extrêmement flexible et souvent très performante.

Nous ne préconisons pas d'adopter systématiquement cette approche. Dans la majorité des cas, ce serait tirer sur des moineaux avec des boulets de canon. Mais que les concepteurs, débutants ou aguerris, se rappellent que, dans les cas où cela se justifie, il existe cette possibilité de décrire le cycle de vie des objets de façon très élégante et efficace.