Vue Materialise

Quelle est une vue matérialisée?

Une vue matérialisée (MV) est le résultat pré-calculée (matérialisée) d'une requête. Contrairement à une vue simple le résultat d'une vue matérialisée est stocké quelque part, généralement dans un tableau. Vues matérialisées sont utilisées lorsque la réponse immédiate est nécessaire et que la requête d'où la vue sur les bases matérialisées prendrait trop longtemps pour produire un résultat. Vues matérialisées doivent être rafraîchi de temps en temps. Cela dépend de la façon dont les exigences souvent une vue matérialisée est rafraîchie et comment réelle de son contenu est. Fondamentalement, une vue matérialisée peut être rafraîchi immédiatement ou à terme, il peut être actualisé en totalité ou un certain point dans le temps. MySQL ne fournit pas vues matérialisées par lui-même. Mais il est facile de construire des vues matérialisées vous-même. Mettre en œuvre vos propres vues matérialisées

Un petit exemple de la façon dont cela pourrait être fait est la requête suivante:

```
SELECT COUNT(*)
 FROM MyISAM_table;
```

renvoie un résultat immédiat parce que le compteur est stockée dans l'en-tête de la table. La requête suivante peut prendre quelques secondes à minutes:

```
SELECT COUNT(*)
FROM innodb_huge;
```

Une solution possible serait de créer une table où tous les chefs de ligne, InnoDB sont stockés dans:

```
CREATE TABLE innodb_row_count (
  id          INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY
 , schema_name VARCHAR(64)  NOT NULL
 , table_name  VARCHAR(64)  NOT NULL
 , row_count   INT UNSIGNED NOT NULL
 );
```

n fonction de la justesse nécessaire de cette information, le tableau peut être actualisée une fois par jour (moins utilisé les ressources du système, mais plus grosse erreur dans le résultat), une fois par heure ou dans le cas le plus extrême, après chaque changement (le plus lent)!

Une autre possibilité serait d'obtenir les données à partir du schéma de l'information. Mais cette information peut être jusqu'à 20% tort!

```
SELECT table_schema, table_name, table_rows  FROM information_schema.tables
WHERE table_type = 'BASE TABLE';
```

Actualiser les vues matérialisées

Vues matérialisées peuvent être actualisées de différentes manière:

- jamais (une seule fois au début, pour des données statiques seulement)
- sur demande (par exemple une fois par jour, par exemple après le chargement de nuit)
- immédiatement (après chaque instruction)

Une actualisation peut se faire par les moyens suivants:

- complètement (lent, plein de zéro)
- différée (rapide, par une table de journal)

En stockant les informations de modification dans une table de journalisation. En outre, certains clichés ou retardées temps, les États peuvent être produites:

- rafraîchir à jour
- rafraîchissement complet

En pratique

Pour comprendre tout cela plus en détail, il est probablement plus facile de faire des exemples. Supposons que nous avons un chiffre d'affaires de table

```
CREATE TABLE sales (

   sales_id       INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY
 , product_name   VARCHAR(128) NOT NULL
 , product_price  DECIMAL(8,2) NOT NULL
 , product_amount SMALLINT     NOT NULL
```

```
);

INSERT INTO sales VALUES

 (NULL, 'Apple', 1.25, 1), (NULL, 'Apple', 2.40, 2)

, (NULL, 'Apple', 4.05, 3), (NULL, 'Pear', 6.30, 2) , (NULL, 'Pear', 12.20, 4), (NULL, 'Plum', 4.85, 3)

SELECT * FROM sales;
```

Et maintenant, nous voulons connaître le prix vendu et l'argent gagné par produit:

```
EXPLAIN

SELECT product_name

    , SUM(product_price) AS price_sum, SUM(product_amount) AS amount_sum
    , AVG(product_price) AS price_avg, AVG(product_amount) amount_agg
    , COUNT(*)
 FROM sales
GROUP BY product_name
ORDER BY price_sum;

+-------------+-------+------+---------------+------+---------------------------------+
|select_type | table | type | possible_keys | rows | Extra |
+-------------+-------+------+---------------+------+---------------------------------+
| SIMPLE | sales | ALL | NULL | 6 | Using temporary; Using filesort |
+-------------+-------+------+---------------+------+---------------------------------+
```

Créer votre propre vue matérialisée

```
DROP TABLE sales_mv; CREATE TABLE sales_mv (

   product_name VARCHAR(128)  NOT NULL
```

```
 , price_sum    DECIMAL(10,2) NOT NULL
 , amount_sum   INT           NOT NULL
 , price_avg    FLOAT         NOT NULL
 , amount_avg   FLOAT         NOT NULL
 , sales_cnt    INT           NOT NULL
 , UNIQUE INDEX product (product_name)

);

INSERT INTO sales_mv SELECT product_name

   , SUM(product_price), SUM(product_amount)
   , AVG(product_price), AVG(product_amount)
   , COUNT(*)
 FROM sales

GROUP BY product_name;
```

C'est jusqu'à présent la partie la plus facile! Et, comme prévu, nous obtenons le résultat correct:

```
 mysql> SELECT * FROM sales_mv;
 +--------------+-----------+------------+-----------+------------+-----------+
| product_name | price_sum | amount_sum | price_avg | amount_avg | sales_cnt
| +--------------+-----------+------------+-----------+------------+-----------+
| Apple | 7.70 | 6 | 2.56667 | 2 | 3 |
| Pear | 18.50 | 6 | 9.25 | 3 | 2 | | Plum | 4.85 | 3 | 4.85 | 3 | 1 |
 +--------------+-----------+------------+-----------+------------+-----------+
 3 rows in set (0.00 sec)
```

Cela couvre le mode de rafraîchissement "JAMAIS" Mais ce n'est pas ce que nous voulons généralement à faire. Actualiser vue matérialisée sur demande

L'actualisation de la vue matérialisée sur la demande peut être mis en œuvre avec une procédure stockée comme suit:

```
ROP PROCEDURE refresh_mv_now;

DELIMITER $$

CREATE PROCEDURE refresh_mv_now (
```

```
   OUT rc INT

) BEGIN

 TRUNCATE TABLE sales_mv;

 INSERT INTO sales_mv
 SELECT product_name
     , SUM(product_price), SUM(product_amount)
     , AVG(product_price), AVG(product_amount)
     , COUNT(*)
   FROM sales
 GROUP BY product_name;

 SET rc = 0;

END; $$

DELIMITER ;
```

Pour vérifier si cela fonctionne la déclaration suivante a été utilisée:

```
CALL refresh_mv_now(@rc);

SELECT * FROM sales_mv;

+--------------+-----------+------------+-----------+------------+-----------+
| product_name | price_sum | amount_sum | price_avg | amount_avg | sales_cnt
| +--------------+-----------+------------+-----------+------------+-----------+
| Apple | 7.70 | 6 | 2.56667 | 2 | 3 |
| Pear | 18.50 | 6 | 9.25 | 3 | 2 |
| Plum | 4.85 | 3 | 4.85 | 3 | 1 |
+--------------+-----------+------------+-----------+------------+-----------+

INSERT INTO sales VALUES

 (NULL, 'Apple', 2.25, 3), (NULL, 'Plum', 3.35, 1)

, (NULL, 'Pear', 1.80, 2);
```

```
CALL refresh_mv_now(@rc);

SELECT * FROM sales_mv;

+--------------+-----------+------------+-----------+------------+-----------+
| product_name | price_sum | amount_sum | price_avg | amount_avg | sales_cnt
| +--------------+-----------+------------+-----------+------------+-----------+
| Apple | 9.95 | 9 | 2.4875 | 2.25 | 4 | | Pear | 20.30 | 8 | 6.76667 | 2.66667 | 3 |
| Plum | 8.20 | 4 | 4.1 | 2 | 2 |
+--------------+-----------+------------+-----------+------------+-----------+
```

Actualiser vue matérialisée immédiate

Pour faire un rafraîchissement complet après chaque déclaration n'a pas de sens. Mais nous aimerions toujours avoir à bon résultat. Pour ce faire, il est un peu plus compliqué.

Sur chaque INSERT sur la table de vente, nous devons mettre à jour notre vue matérialisée. Nous pouvons mettre en œuvre cette manière transparente par INSERT / UPDATE / DELETE déclencheurs sur la table sales:

Maintenant, laissez-nous créer les déclencheurs nécessaires:

```
 DELIMITER $$

CREATE TRIGGER sales_ins AFTER INSERT ON sales FOR EACH ROW BEGIN

 SET @old_price_sum = 0;
 SET @old_amount_sum = 0;
 SET @old_price_avg = 0;
 SET @old_amount_avg = 0;
 SET @old_sales_cnt = 0;

 SELECT IFNULL(price_sum, 0), IFNULL(amount_sum, 0), IFNULL(price_avg, 0)
      , IFNULL(amount_avg, 0), IFNULL(sales_cnt, 0)
   FROM sales_mv
  WHERE product_name = NEW.product_name
   INTO @old_price_sum, @old_amount_sum, @old_price_avg
```

```
      , @old_amount_avg, @old_sales_cnt
 ;

 SET @new_price_sum = @old_price_sum + NEW.product_price;
 SET @new_amount_sum = @old_amount_sum + NEW.product_amount;
 SET @new_sales_cnt = @old_sales_cnt + 1;
 SET @new_price_avg = @new_price_sum / @new_sales_cnt;
 SET @new_amount_avg = @new_amount_sum / @new_sales_cnt;

 REPLACE INTO sales_mv
 VALUES(NEW.product_name, @new_price_sum, @new_amount_sum, @new_price_avg
      , @new_amount_avg, @new_sales_cnt)
 ;

END; $$

DELIMITER ;
```

Revision #2
Created 29 September 2022 04:56:46 by ggpilou2
Updated 29 September 2022 08:00:47 by ggpilou2