12. 视图与索引

12.1 视图

视图 (view)：一个定义关系但并不实际（在物理层面）创造关系的查询。对于视图的查询以及修改与正常表非常类似。

视图与基本表的关系

- **基本表** (base table)： 是在物理层面存储的表，用 `CREATE TABLE` 定义；
- **导出表** (derived table)：从一个或多个其他表中使用类似查询的命令导出；
- **视图**：是一个命名的导出表，并没有在物理层面进行存储，在基本表上定义，用 `CREATE VIEW` 定义。

视图的作用

- 对用户隐藏部分数据；
- 让部分查询可以更简单、自然的表述；
- 模块化。

视图的创建：使用语句 `CREATE VIEW viewname(A1, A2, ..., An) AS query;` 将 `query` 查询的结果定义为视图。在系统内部，只存储视图对定义，而没有实际的表来存储视图（即并没有物理层面存储查询的结果）。例如：

```
CREATE VIEW CS_S AS
 SELECT sno, name, age
 FROM S
 WHERE dept='CS';
```

视图的查询：在用户层面，可以将视图当成正常存储的表进行查询。在视图上的查询会被翻译为对于基本表的查询，即使用子查询将视图利用定义进行替换即可。

- 可以在视图中对属性进行重命名，如下所示：

  ```
  CREATE VIEW CS_S(id, sn, sa) AS
    SELECT sno, name, age FROM S WHERE dept='CS';
  ```

- 可以创建视图的视图，如下所示：

  ```
  CREATE VIEW CS_S_20 AS
    SELECT sno, name FROM CS_S WHERE age=20;
  ```

视图的修改：一般来说，由于视图并不是实际储存的表格，对视图进行修改没有意义。

- **可更新的视图**：对于非常简单的视图，可以将视图的修改转换至原表格的修改。这点在 SQL 标准中有复杂的规则。

- 视图只能 `SELECT` 属性，而不能使用 `DISTINCT`；而且需要足够多属性（包含基本表的非空属性）使得其能够被插入基本表中；
- 只能从一个关系 `R` 中选取（不过 `R` 也可以是可更新的视图）；
- `WHERE` 后不能跟包含 `R` 的子查询（嵌套）。

例如，`CS_S` 是可更新的视图，但是有重名，如：

```
INSERT INTO CS_S VALUES ('007', 'james', 18);
```

上述代码会将 `('007', 'james', 18)` 加入基本表 `S` 中，但是由于这个元组的 `dept = NULL`，其在基本表中不符合筛选视图 `CS_S` 的条件，故实际上 `CS_S` 没有变化。

- **Instead-of 触发器**：使用 Instead-of 触发器可以用来将视图的修改转换为原表格的修改。数据库设计者可以编写触发器来对视图的修改进行正确的解析，如：

```
CREATE TRIGGER insertCSS
INSTEAD OF INSERT ON CS_S
REFERENCING NEW ROW AS nr
FOR EACH ROW
  INSERT INTO S VALUES(nr.sno, nr.name, nr.age, 'CS');
```

视图的删除：使用 `DROP VIEW viewname;` 来删除视图。注意视图的删除仅仅删除视图的定义，基本表不受到任何影响；与之对应的，`DROP TABLE` 则会影响表与定义在其上的视图。

物化视图：视图在数据库中仅储存其定义，但如果视图经常被应用到，使用物化视图（也就是将视图存在数据库中）将会更加高效：

- **物化视图的创建**：在创建时加入 `MATERIALIZED` 标识，如下所示：

```
CREATE MATERIALIZED VIEW CS_S AS
  SELECT sno, name, age FROM S WHERE dept='CS';
```

- **物化视图的维护**：问题在于每当基本表被修改后，其上的物化视图需要被重新计算，开销大。
  - 数据库可以通过计算基本表修改量的方法计算物化视图相应的修改量进行维护，不过当元组插入/删除时，物化视图的改变并不一定直观。所以现在来看，物化视图的维护基本舍弃了当场维护。
  - **周期性维护**：创建物化视图，并以一个特定周期维护物化视图，使得物化视图的维护工作量可以接受。数据可能会过时，不过风险较低可以接受（物化视图主要用于数据分析领域，个别数据的结果对统计影响较小）。
  - 借助物化视图来重写查询：若有物化视图定义如下：

    ```
    SELECT LV FROM RV WHERE CV;
    ```

    且有查询定义如下：

    ```
    SELECT LQ FROM RQ WHERE CQ;
    ```

如果 $R_V \subseteq R_Q, C_Q = C_V \text{ and } C$；且如果 C 中提及了 R_V 的属性 A，那么 $A \subseteq L_V$；同时 $(L_Q \cup R_V) \subseteq L_V$，那么可以将查询使用物化视图重写如下：

```
SELECT LQ FROM V, RQ-RV WHERE C
```

- **物化视图的自动创建**
  - 首先，我们需要估计查询的负担（哪些查询用的比较多）；
  - 其次，一个自动物化视图选择建议器会提供物化视图候选者，我们可以选择那些负载最大的查询创建物化视图；
  - **查询优化器**会估计创建物化视图和不创建物化视图所需要的时间和空间负担，如果创建物化视图更有利，则会创建之（这里的时间和空间负担包括速度与空间负担，比如一些涉及到 join 的物化视图非常大从而创建开销过大）。

12.2 索引

索引：关系代数的索引是为加速查询某些通过属性 A 筛选出的元组而产生的；A 被称为**属性键 (index key)**。

- 属性键可以是任何元组或元组的集合，不一定需要是关系 R 的键值。
- 使用 B 树或 B+ 树来实现索引。
- **优点**：加速查询，特别对于 `R.A = value` 与 `R.A <= value` 非常有效；同时加速连接 (join)，可以使用索引来找到配对元组 (matching tuples)。
- **缺点**：由于修改时索引也需要被修改，因此使用索引会降低修改时的效率。
- **设计时需要考虑的问题**：正确选择创建什么属性的索引。

索引的创建/删除：索引不是 SQL 标准的一部分，不过大多数商用数系统支持数据库设计者进行索引创建。可以使用 `CREATE INDEX index_name ON index_key` 创建索引并使用 `DROP INDEX index_name` 来删除索引。

```
CREATE INDEX nameIdx ON S(name);
CREATE INDEX scIdx ON SC(sno, cno);
DROP INDEX nameIdx;
```

索引的选取

- **有用的索引**
  - 关系的键值：询问经常涉及，同时经常作为返回值返回，而且在连接中经常使用；
  - 几乎是关系键值的属性，比如：`name`；
  - **聚簇索引**：将某些相关联的元组（如 `dept` 属性上相同）分配在连续的页中来减少读取页的开销。
- 为创建最好的索引，我们需要数据库中关于查询与修改的信息（哪些查询/修改比较可能发生之类）。这个可以从用户历史或应用代码进行分析。
- 下面是一个分析索引的例子

SC(sno,cno)

Q_1: SELECT cno FROM SC WHERE sno='xxx';

Q_2: SELECT sno FROM SC WHERE cno='yyy';

I: INSERT INTO SC VALUES('xxx','yyy');

Prob(Q_1) = p_1, Prob(Q_2) = p_2, Prob(I) = p_3 =1 − p_1 − p_2

SC occupies n pages.

On the average, 'xxx' takes c courses and 'yyy' is taken by s students.

action	no index	index on sno	index on cno	both index
Q_1	n	1+c	n	1+c
Q_2	n	n	1+s	1+s
I	2	4	4	6
Average	$np_1+np_2+2p_3$	$(1+c)p_1+np_2+4p_3$	$np_1+(1+s)p_2+4p_3$	$(1+c)p_1+(1+s)p_2+6p_3$

根据例子通过不同的 p_1, p_2 以及 s, c 来判断需要创建哪些索引。

自动索引选择：数据库的调整建议器，会根据查询负载、限制来找到索引候选者，然后索引选择器将会通过计算每个索引的代价来建议创建索引（或直接创建）。

- **贪心索引选择**：按顺序评估索引，首先创建一个任意索引；然后评估建立其他索引会不会更优，如果会则更新。在评估过程中保持原有索引的创建状态以加快访问速度。