

Views

Useful SQLite Commands

- *.eqp on|off*
 - Show execution plan for SQL query
- *.schema*
 - Show CREATE [TABLE & VIEW] statements
- *.tables*
 - Show tables and views

Virtual Views

- The equivalent of macros and inline functions from C/C++
 - #define constructs, functions in header files
 - Give a name to a code segment rather than copy the code in multiple places
 - The copying is done automatically by the macro processor or compiler without the programmer intervention
- Improve code organization and readability
- No performance benefit

Virtual View Definition in SQL

```
CREATE VIEW Printer_Maker(model, color,  
type, price, maker) AS
```

```
    select Pr.model, Pr.color, Pr.type, price, maker  
    from Printer Pr, Product P  
    where Pr.model = P.model
```

```
DROP VIEW Printer_Maker
```

View Usage in SQL Queries

```
select *  
from Printer_Maker
```

```
select *  
from  
    (select Pr.model, Pr.color,  
        Pr.type, price, maker  
    from Printer Pr, Product P  
    where Pr.model =  
        P.model) Printer_Maker
```

View Usage in SQL Queries

```
select distinct maker  
from product p,  
Printer pr  
where p.model =  
pr.model
```

```
and color = true  
and price < 200
```

```
select distinct maker  
from Printer_Maker  
where color = true  
and price < 200
```

Virtual Views

- The SQL query in the view definition is not evaluated, it is simply given a name *Printer_Maker*
- In SQLite, *CREATE VIEW* is added to the existing database tables
 - *.tables* or *.schema*
 - There is no other change to the database
- The query execution plans with and without the view are exactly the same
 - *.eqp on*

Modification Operations on Tables in the Virtual View Definition

- DROP TABLE Printer
 - View *Printer_Maker* becomes invalid
- INSERT/DELETE/UPDATE on Printer
 - All modification operations are immediately reflected in the view since the query in the view is re-evaluated every time it is included in a query
 - Exactly the same behavior as for tables

Modification Operations on Virtual Views

INSERT INTO Printer_Maker(model, color, type, price, maker)

VALUES(3108, false, 'laser', 169, 'A')

- INSERT INTO Printer(model, color, type, price)
VALUES(3108, false, 'laser', 169)
- INSERT INTO Product(model, type, maker)
VALUES(3108, 'printer', 'A')

Modification Operations on Virtual Views

```
CREATE VIEW Prod_Printer(model, maker) AS  
    SELECT model, maker  
    FROM Product  
    WHERE type = 'printer'
```

```
INSERT INTO Prod_Printer(model, maker)  
VALUES(3108, 'A')
```

- INSERT INTO Product(model, type, maker)
VALUES(3108, NULL, 'A')
- SELECT * FROM Prod_Printer
 - **(3108, 'A') is not in the result**

Modification Operations on Virtual Views

- **Not supported in SQLite**
- SQL standard defines **UPDATABLE VIEWS**

```
CREATE VIEW Prod_Printer(model, maker, type)  
AS
```

```
    SELECT model, maker, type
```

```
    FROM Product
```

```
    WHERE type = 'printer'
```

Materialized Views

- Query result caching (materialization) into a table
- Use in queries exactly as tables or views
- Avoid recomputation by returning result directly
- Related to memoization from dynamic programming
- **Improve query performance**

Materialized View Definition in SQL

CREATE MATERIALIZED VIEW

Printer_Maker_M(model, color, type, price,
maker) AS

select Pr.model, Pr.color, Pr.type, price, maker
from Printer Pr, Product P
where Pr.model = P.model

DROP MATERIALIZED VIEW Printer_Maker_M

- **Not supported in SQLite**

Simulate Materialized Views in SQLite

CREATE MATERIALIZED VIEW

```
Printer_Maker_M(model,  
color, type, price, maker)  
AS
```

```
select Pr.model, Pr.color,  
Pr.type, price, maker  
from Printer Pr, Product P  
where Pr.model = P.model
```

• **CREATE TABLE**

```
Printer_Maker_M(model,  
color, type, price, maker)
```

• **INSERT INTO**

```
Printer_Maker_M
```

```
SELECT Pr.model,  
Pr.color, Pr.type, price,  
maker
```

```
from Printer Pr, Product P  
where Pr.model = P.model
```

Modification Operations on Materialized Views

- Since the materialized view is a table, I/U/D operations are straightforward
- View is not consistent with its definition anymore
- **For consistency, modification operations have to be propagated to the base tables in the view definition**
 - Same approach as for virtual views

Materialized View Maintenance

- Materialized view is a separate table
 - Independent copy of data
- Modification operations on tables in the view definition have to be propagated to the view
 - **View is consistent with base tables**
- Naive materialized view maintenance
 - Complete reevaluation
 - DELETE FROM Printer_Maker_M
 - INSERT INTO Printer_Maker_M (SELECT ...)

Incremental View Maintenance

- Minimize the number of tuples from the materialized view that get impacted by a modification operation on base tables
- Implemented for every I/U/D operation separately
 - Consider only modified tuples from base tables

INSERT Product+Printer

- INSERT INTO Product(model, type, maker)
VALUES(**3108**, 'printer', 'A')
- INSERT INTO Printer(model, color, type, price)
VALUES(**3108**, false, 'laser', 169)
- **INSERT INTO Printer_Maker_M**
*(SELECT Pr.model, Pr.color, Pr.type, price, maker
from Printer Pr, Product P
where Pr.model = P.model AND P.model = 3108)*

DELETE Printer

- DELETE FROM Printer WHERE model < 3004
- **DELETE FROM Printer_Maker_M
WHERE model < 3004**

UPDATE Product

- UPDATE Product
SET maker = 'A'
WHERE maker = 'D'
- **UPDATE Printer_Maker_M**
SET maker = 'A'
WHERE maker = 'D'

Views Summary

- Virtual views
 - Name for SELECT statement
 - Improve coding
 - No modification operations
 - No query execution performance improvement
- Materialized views
 - Save query result into a separate table
 - Query result caching
 - Always improve query execution performance
 - Incremental view maintenance