

**CS4221 Database Design Project Report**

Title: View Updates on Relational Databases

|  |  |  |
| --- | --- | --- |
| **Group Members** | **Name** | **Matriculation Number** |
| LAU WEI LUN | U096892W |
| SNG AIK WEI | U096884X |
| WONG KANGWEI | U096904B |

1. Introduction

Views in relational databases are external schema and are virtual tables representing the result set of a database query. It offers the flexibility of manipulating the representation of collated data in order to extract information that are relevant to the user, and at the same time, ignore data that are irrelevant to him.

Views can have added capability by allowing users to apply update operations to it; however, there exist concerns regarding procedures which involve translating changes made in virtual representation to physical representation. Treatment of such translations has been heavily discussed in academics but for the most part has been extremely ad hoc as discussed in (Date C. J., 1986), as such, it is the aim of this project to recognize treatments that are correct and implement these methods to supplement an existing relational database management system that does not currently support updatable views.

To achieve this purpose, we have chosen the relational language SQL due to its support for primary and foreign keys for reasons mentioned later in this report. In accordance with (Date C. J., 1986), which describes itself as a proposal for view updates for SQL, we have adopted a similar framework in our implementation of view updates for the chosen DBMS - SQLite.

It is not our intention to provide a complete treatment of the subject due to constraints imposed on this project. While the project limits itself to *restriction*, *projection* and *join* views, it is believed that the scope of this project may be extended to include other functionalities.

1. SQLite

SQLite prides itself as “a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine”. In this project, we chose to supplement SQLite with updatable views as it implements SQL92 which supports primary and foreign keys, as well as the ability to cater to restriction, projection and join views despite lacking the ability to update views.

Its increased use in the industry, particularly in embedded systems such as Google’s Android[[1]](#footnote-1), also has minor influence on our choice. It is of our hope that as a byproduct, our project helps users of SQLite circumvent the inability to write to views.

1. Selection criteria for treatment method

As mentioned in the above sections, while there has been intensive study on the topic of updatable views, most are considered ad hoc. This poses a significant challenge in our aim to sieve out methods which are correct. However, a good selection of papers generally agree on a set of methods and have developed theories based on different frameworks but are generally cohesive.

The criteria imposed on our selection generally include, but is not limited to, the following:

1. The method does not cause us to be encumbered by theoretical conceptions, that is, the method must have been conjured with implementation in mind.
2. The theories used to derive the method have been proven for correctness.
3. The method developed should be within the framework of the relational model and should preferably characterize the conditions of which a change made in virtual representation can be translated to a physical representation.
4. Survey on existing research

(Date., 1995) characterizes the conditions necessary to satisfy insert, deletion and replacement operations by considering whether the operated row satisfies its corresponding table predicates and restriction conditions (if any).

(Ling & Lee, 1996) presents view update within the Entity Relationship approach which characterizes the conditions under which there are mappings from view updates for view entity types and view relationship sets into updates on the physical representation. While such an approach differs from (Date., 1995) in the sense that they pertain to different frameworks, results arising from his research is also applicable to the relational model and can be treated as “a special case of the ER approach”.

(Date C. J., 1986) presents a discussion of the view update problem in the context of a query language with the assumption that the language in question provides full support for primary and foreign keys. It mentions that updatability is a semantic issue which requires knowledge of the meaning of the data as well as primary and foreign keys. Using the relational language SQL, it demonstrates updatability of a view through consideration table predicates that must be satisfied and the subsequent operations supported by SQL to be performed.

1. Theory

In light of our selection criteria, we have chosen to adapt view update theories discussed in (Date C. J., 1986), (Date., 1995) and (Ling & Lee, 1996). The following section briefly discusses these theories and implementation details.

* 1. **Preliminaries**

When the term “table predicates” is mentioned, it refers to the associated meaning of a base table, a view or a query result. [Updating Union Intersection and Difference Views] provides an example of what is called a predicate or truth-valued function:

*“The employee with the specified employee number (EMP#) has the specified name (ENAME) works in the specified department (DEPT#) and earns the specified salary (SAL). Furthermore, if the department number is D1, then the salary is less than 44K. Also, no two employees have the same employee number.”*

Substituting values for the arguments is akin to instantiating the predicate which either yields *true* or *false.* For example **EMP# = ‘E1’ ENAME=‘Lopez’ DEPT#=‘D1’ SAL=25K** yields the value *true* and **EMP# = ‘E1’ ENAME=‘Lopez’ DEPT#=‘D1’ SAL=45K** yields *false*. It follows that in the case where a row is presented as a candidate for insertion into some table, the DBMS should accept that row only if it does not cause the corresponding predicate to be violated. The criterion for update acceptability for a table is thus represented by its table predicate. (Date., 1995) presents the remaining principles as follows:

1. The table predicate for a given table constitutes the criterion for deciding whether or not some proposed update is in fact valid for that table.
2. In the case of a base table, the table predicate is logical AND of all column constraints and table constraints that apply to the base table in question.
3. In the case of a derived table, the table predicate is derived in a straightforward way from the table predicate(s) for the table(s) from which the table in question is derived; for example, the table predicate PC for C = A B is (PA) and (PB), where PA and PB are table predicates for A and B, respectively.
4. The view updatability rules must work correctly in the special case when the “view” is in fact a base table.
5. It also follows from (Ling & Lee, 1996) that in the Entity Relation framework, a “view entity type” is insertable if and only if the identifier of its base entity type is included in the view, (Date C. J., 1986) reinforces this by stating that “a table that does not have a primary key is not updatable”.

In this and the following sections we cover the issue of updatability of restrictions and projections and also join views respectively, which is mainly due from (Date C. J., 1986)and also their implementation details using the SQL framework. While implementation details for updating join views cover both inner and outer joins (where outer joins are beyond the scope of this project), we shall also include the treatment of outer join views here to cater to the possibility of deferred implementation in future.

* 1. **Updating Restrictions**

Informally, a restriction is always updatable where updates are simply mapped directly into updates on the underlying relation. By (Date C. J., 1986), the treatment of such an update should ensure that the resulting relation continues to satisfy the restriction predicate. Specifically, in the case where a restriction operation results in a table predicate (*PA*) AND (*condition*), the update rules for A WHERE condition is as follows:

**INSERT:** The candidate row must satisfy both *PA* and *condition*.

**DELETE:** The row to be deleted is deleted from A.

**UPDATE:** The row to be updated must be such that the updated row satisfies both *PA* and *condition*.

**Updating Projections**

By (Date C. J., 1986), a projection is updatable if and only if it preserves the primary key of the underlying relation, only then can the system determine the precise tuple to be updated in that underlying relation. Specifically, in the case when A[X] is the projection of A where A is partitioned into two disjoint groups, X and Y, a given row <*x*> appears in A[X] if and only if there is an existing y value such that the row <*x,y*> appears in A, the update rules for A[X] are as follows:

**INSERT:** Let the candidate row be <x> and y the default value (i.e. NULL; note that it is an error if Y has defaults not allowed). If the row <x,y> satisfies PA then it is inserted into A.

**DELETE:** All rows of A with the same X-value as the row to be deleted from A[X] are deleted from A.

**UPDATE:** Let the row to be updated be <*x*> and the updated version be <*x’*>. Let *a* be a row of A with the same X-value *x*, and let the value of Y in row *a* be *y*. All such rows *a* are updated to <*x’,y*> if they satisfy PA.

* 1. **Updating Joins (PK,PK)-JOIN**

We consider only the case in which the attributes over which the attributes over which the join is taken are the primary key R.Rp of the one relation and primary key S.Sp matching R.Rp in the other relation (S).

Updates on a (PK,PK)-join map to corresponding updates to both of the participant relations. For instance, deleting a tuple from such a join maps to a delete on both the relations.

Specifically, let the columns of table A be partitioned into two disjoint groups X and Y. Also, let the columns of table *B* be partitioned into two disjoint groups Y and Z. Suppose the columns of Y (only) are common among both A and B, then A JOIN B yields a table with columns {X,Y,Z} consisting of rows <*x,y,z*> where <*x,y*> appears in A and <*y,z*> appears in B. It then follows that the table predicate PJ for J = A JOIN B is PA *(a)* and PB *(b)* where for a given row *j* of the join, *a* is the “A-part” of *j* (the row derived from *j* by projecting away the value *j*.Z) and b is the “B-part” of *j* (derived from *j* by projecting away the value *j*,X). In other words: “Every row in the join is such that the A-portion satisfies PA and the B-portion satisfies PB.” Considering such a J, the update rules are as follows:

**INSERT:** The new row *j* must satisfy PJ. If the A-part of j does not appear in A, it is inserted into A. If the B-part of *j* does not appear in B, it is inserted into B.

**DELETE:** The A-part of the row to be deleted is deleted from A and the B-part is deleted from B.

**UPDATE:** The update to a row in J must satisfy PJ.

1. System Architecture

CLI

SQLite DB

Update View Controller

Parser

Query Processor

DB Connector

……

……

**Step 1:** Command Line Interface parses SQL query by user to Update View Controller.

**Step 2:** View Update Controller sends the SQL query to the Parser.

**Step 3:** The parser analyses SQL query and captures the type of SQL query, i.e. insert, delete, update or create and checks for projection, restriction and join operation. It also interprets the SQL query restriction and represenst it as abstract syntax trees. In addition, it keeps track of physical databse table’s mapping with corresponding projected columns of the view. All these information will be saved and returned to View Update Controller.

**Step 4:** View Update Controller parses the information generated by Parser to Query Processor to determine if the query is updatable. Query Processor will check if the query fulfills the table predicate and restriction constraints.

**Step 5:** If SQL query does not violate any table predicate or restriction constraint, Query Processor will transform view updates into physical table updates by performing the necessary operations for view updates using DB Connector. If the SQL query does not fulfill the requirement, no operation will be performed. Query Processor will then notify View Update Controller the result of the operations.

**Step 6:** View Update Controller will notify Command Line Interface whether the operation is successful.

1. User Guide

## 7.1. SQL Syntax

We have implemented a subset of the SQL language which emulates its behaviour as closely as possible. In the following sections of this short user guide, we have supplemented the relevant sections with SQL format and examples. Take note of the following notations in the following sections:

* Angle brackets [ ] are segments of SQL syntax which are optional.
* Arrow brackets < > are not part of the SQL syntax
* Words in upper case are reserved keywords of SQL, for e.g. SELECT, WHERE, etc.
* Words in lower case may refer to column names, values, view names, or table names.
* Every SQL query has to end with a semi-colon ;
* Values are represented as integer constants, e.g. 100, 128, etc.
* Strings are enclosed with single-quotations, e.g. ‘Chan Mali Chan’, ‘Tan Ah Kow’, etc.

## 7.2. Defining Views

In this project, we have implemented projections, restrictions and PK-PK inner joins on database views. The database administrator may use any combinations of such to build his views of interest. White spaces are not allowed when defining names of views.

### 7.2.1. Projection and/or Restriction

To define views on a single physical table, enter the SQL command according to the syntax as follows:

CREATE VIEW view\_name AS SELECT <projection> FROM physical\_table

[WHERE <restriction>];

If there is no projection required, then replace <projection> with the asterisk \* to refer to all columns. To define a projection to the view, separate each column name with a comma , . For example, emp\_id, name, address, salary.

The optional WHERE clause is used to extract only tuples that fulfill a certain criterion, i.e. the restriction of the view. The following operators can be used in restriction:

|  |  |
| --- | --- |
| Operator | Description |
| = | Equal |
| != | Not equal |
| > | Greater than |
| >= | Greater than or equal |
| < | Less than |
| <= | Less than or equal |
| AND | True only if both the first and second condition is true. Else false. |
| OR | True if either the first or the second condition is true. Else false. |

Additionally, the following arithmetic operators can be used in restriction:

|  |  |
| --- | --- |
| Operator | Description |
| + | Addition |
| - | Subtraction |
| \* | Multiplication |
| / | Division |
| % | Modulo (Remainder of division) |

### 7.2.2. PK-PK Inner Join

To define views for PK-PK inner-joins, enter the SQL command according to the syntax as follows:

CREATE VIEW view\_name AS SELECT <projection> FROM phyTable1

INNER JOIN phyTable2

ON phyTable1.id = phyTable2.id

[WHERE <restriction>];

The ON clause specifies the predicate for the inner join. The view will join physical table 1 phyTable1 and physical table 2 phyTable2 using the primary key id of both tables. Restriction and projection may also be defined on join views. When specifying projection for PK-PK inner-joins, table name must be prepended to the column name. For example, table1.column1, table1.column2, table2.column1.

## 7.3. Dropping Views

You can drop a view using the following SQL command:

DROP VIEW view\_name;

View will be removed from the database.

## 7.4. Querying Views

To query a particular view, enter the SQL command according to the syntax as follows:

SELECT <column\_name(s)> FROM view\_name [WHERE <restriction>];

Column names are seperated by comma , . For example, column1, column2, column3… Column names can also be replaced by asterisk \* to indicate the selection of all columns. Restriction may be defined when querying views.

## 7.5. Insertions

To insert a tuple into the view, enter the SQL command according to the syntax as follows:

INSERT INTO view\_name (<column1, column2, column3,…>)

VALUES (<value1, value2, value3,…>)

A tuple will be inserted based on the mapping between column name(s) and the corresponding value(s). For example, value1 will be inserted to column1 and value2 will be inserted to column2 for the particular tuple.

## 7.6. Deletions

To delete tuple(s) from the view, enter the SQL command according to the syntax as follows:

DELETE FROM view\_name [WHERE <restriction>];

Restriction may be defined on deletions. If no restriction is indicated, all tuple in the view will be deleted.

## 7.7. Updates

To update tuple(s) in the view, enter the SQL command according to the syntax as follows:

UPDATE view\_name SET <column1=value1, column2=value2,…>

[WHERE <restriction>];

Tuples matching the restriction criterion will have the corresponding value that maps to the column name updated. For instance, the value of column1 of a particular tuple will be updated to value1. Restriction (WHERE clause) may be defined on a update query. If no restriction is indicated, all tuples in the view will have their columns updated to their corresponding values. For instance, value of column1 will be updated to value1.

# Bibliography

Date, C. (1995). Updating Union, Intersection, and Difference Views. In C. Date, *Relational database writings, 1991-1994* (pp. 251-266). Boston, Massachusetts: Addison-Wesley.

Date, C. J. (1986). Updating Views. In C. J. Date, *Relational Database: Selected Writings* (pp. 367-395). Boston, Massachusetts: Addison-Wesley Pub. Co.

Date., C. (1995). Updating Joins and Other Views. In C. Date., *Relational database writings, 1991-1994* (pp. 267-283). Boston, Massachusetts: Addison-Wesley.

Wang, L. T., & Li, L. M. (1996). View update in entity-relationship based database management systems. *Data & Knowledge Engineering Vol.19, Issue 2*, 135-169.

1. an operating system for smartphones and tablets [↑](#footnote-ref-1)