ECE356: Database Systems Winter 2018

Lecture 5 — More SQL

Jeff Zarnett

SQL: the Sequel

With some theoretical understanding of the relational algebra that makes up the basis for SQL we can now take some time to learn a little more of the practical ins and outs of the language that will allow us to do the things we need our database to do.

String Operations

Null Values

Aggregate functions Grouping/Having

Set Comparison

Unique

Modification

Until now all the operations have been of the look-but-don't-touch variety – we have looked at the data, combined it, sliced and diced it, and generated some temporary values – but that's it; we haven't actually changed the data in any permanent way. There are three basic operations we can do: add, remove, and change.

Keep in mind that modification operations are transactions (that is to say, they are supposed to succeed or be as if it never started). So what really happens is that a new relation is created and that replaces the original relation. These operations can also be expressed using relational algebra with an assignment.

Insert. The insert statement creates a new tuple and adds it to the relation, or creates a set of tuples and then adds that set to the relation.

If we are creating just one, then the simple, manual approach is sufficient. In SQL the keywords are `INSERT INTO` and `VALUES`. To add a new license plate we would write: `INSERT INTO license_plate( number, expiry, owner_address_id ) VALUES( 'HOLMES01', '2018-12-10', 86753 );`

Observations: we specify the relation and then we give a list of attributes that we want to assign, followed by the values that we wish to be assigned. In some situations we can leave off the parenthesis-enclosed list of attributes but that is not recommended. The SQL server will try to carry out the assignment of values in the order that the attributes are defined in the database and this may result in undesired behaviour. I therefore insist that the use of the order of attributes is important.

Furthermore, we can put the attributes to be inserted in any order as long as the value types match: `INSERT INTO license_plate( expiry, number, owner_address_id ) VALUES( '2018-12-10', 'HOLMES01', 86753 );` is completely equivalent to the previous statement.

It is permissible to leave some attributes out of the insert statement. `INSERT INTO license_plate( expiry, number ) VALUES( '2018-12-10', 'HOLMES01' );` would create a tuple where the owner address id attribute contains null. This must, however, be permitted by the table definition: i.e., the field must allow null attributes or have a defined default (a boolean attribute, for example, may have "default false" as part of its definition).

We may also insert multiple elements into a relation based on the result of a select statement. `INSERT INTO`

`owner_address ( SELECT * FROM employee_address ) );` is a minimal statement that takes the result of the selection and uses it as a set of tuples to insert. This statement leaves off the attribute specification (which is possible, but not recommended) and has no predicate (where clauses) so it is a rather dangerous statement to write. These sorts of statements are hopefully rare occurrences in your database because they have a risk of going wrong or duplicating data. But sometimes, when, for example, two tables are being merged, they are unavoidable.

In relational algebra, if the relation is r and the new tuples are t, then the insert statement is $r \leftarrow (r \cup t)$.

Delete. The delete operation can only be used to remove a set of tuples from a single relation; it does not alter attributes in the tuples [SKS11]. If the goal is to "clear fields" the update statement is appropriate. A delete statement cannot be used on multiple relations at once.

The keyword in SQL is `DELETE`[1] and the statement `DELETE FROM license_plate where number = 'BBCC 394';` will remove from the license plate relation any and all tuples that match the predicate. If no predicate is specified, then ALL tuples in the relation are removed (yikes!).

Like an insertion we may use the result of a subquery as the predicate in a where clause, so we could delete all license plates where the registered address province is not Ontario with something like `DELETE FROM license_plate WHERE owner_address_id in (SELECT id FROM OWNER_ADDRESS WHERE province <> 'ON');`

The textbook [SKS11] has an example about deletion where the salary is less than the average. This highlights that the deletion operation first figures out the average, then decides what tuples to delete, before carrying out the operation. If this were not the case, after each deletion the average might change and we might delete all tuples (or the result would otherwise be strange). If it carried out the deletion in such a manner it would also break the rule we have about this being a transaction: the tuples to be deleted should be removed in one swift stroke with no partially completed state visible.

In relational algebra then the relation r is modified by a deletion with predicate p as follows: $r \leftarrow (r - \sigma_p(r))$.

Deletion should be used very carefully. When data is removed from the database, it's gone and not coming back (except, well, from backups... You do take backups, right?). Sometimes rather than actually deleting things you may be well advised to "deactivate" them, whether by having a boolean attribute for "deactivated" or having a second relation as a sort of "recycle bin"/"trash can" where tuples to be removed will eventually go.

Update. The update statement changes one or more attributes of the tuple without changing all the values in the tuple.

We could model this as simultaneous deletion of the old tuple and insertion of the new one.

References

[SKS11] Abraham Silberschatz, Henry F. Korth, and S. Sudarshan. *Database System Concepts, 6th Edition*. McGraw Hill, 2011.

[1] `https://www.youtube.com/watch?v=4ecWDo-HpbE`