Lecture 6

SQL DATA MANIPULATION

by

David Toman

University of Waterloo
Fall '05

List of Slides

How do we Modify a Database?

- Naive approach:

  DBSTART;
  $r_1 := Q_1(DB)$;
  ...
  $r_k := Q_k(DB)$;
  DBCOMMIT;

- Not acceptable solution in practice
  $\Rightarrow$ too much copying

Incremental Updates

Idea: Tables are large but **updates are small**:

- Incremental updates:
  1. insertion of a tuples (**`INSERT`**)
     $\Rightarrow$ constant tuple
     $\Rightarrow$ results of queries
  2. deletion of tuples (**`DELETE`**)
     $\Rightarrow$ based on match of a condition
  3. modification of tuples (**`UPDATE`**)
     $\Rightarrow$ allows updating “in place”
     $\Rightarrow$ based on match of a condition

SQL Insert

- one constant tuple (or a fixed number):

```
INSERT INTO r[(a1,...,ak)]
       VALUES (v1,...,vk)
```

  $\Rightarrow$ adds tuples $(v_1, \ldots, v_k)$ to r.
  $\Rightarrow$ the type of $(v_1, \ldots, v_k)$
  must match the schema definition of r.

- multiple tuples (generated by a query) Values generated by a query:

```
INSERT INTO r ( Q )
```

  $\Rightarrow$ adds result of Q to r

Example: inserton of a tuple

Add a new author:

```
SQL> insert into author
  2         values (4, 'Niwinski, Damian',
  3            'http://zls.mimuw.edu.pl/~niwinski' )

1 row created.

SQL> select name,url from author;

AID NAME             URL
--- ---------------  ------------------------------
  1 Toman, David     http://db.uwaterloo.ca/~david
  2 Chomicki, Jan    http://cs.monmouth.edu/~chomick
  3 Saake, Gunter
  4 Niwinski, Damian http://zls.mimuw.edu.pl/~niwins
```


Example: use of a query

Add a new author (without looking up author id):

```
insert into author (
       select max(aid)+1, 'Snodgrass, Richard T.',
              'http://www.cs.arizona.edu/people/rts'
       from author
       )
1 row created.

SQL> select aid, name from author;

AID NAME
--- ----------------------
  1 Toman, David
  2 Chomicki, Jan
  3 Saake, Gunter
  4 Damian Niwinski
  5 Snodgrass, Richard T.
```

SQL Delete

- Deletion using a condition:

```
DELETE FROM r
WHERE cond
```

  ⇒ deletes **all** tuples that match **cond**.

- Deletion using cursors (later)

  ⇒ available in embedded SQL
  ⇒ only way to delete one out of two duplicate tuples

Example

Delete all publications that are not articles or the collections an article appears in:

```
SQL> delete from publication
  2         where pubid not in (
  3               select pubid
  4               from article
  5         ) and pubid not in (
  6               select crossref
  7               from article
  8         )

0 rows deleted.
```

SQL Update

- Two components:
  1. an update statement (**SET**)

     $\Rightarrow$ an assignment of values to attributes.
  2. a search condition (**WHERE**)
- Syntax:

```
UPDATE r
SET    <update statement>
WHERE  <condition>
```


Example

```
SQL> update author
  2  set    url = 'http://brics.dk/~david'
  3  where  aid in (
  4           select aid
  5           from author
  6           where name like 'Toman%'
  7*        )

1 row updated.

SQL> select * from author;

AID NAME            URL
--- --------------- ------------------------------
  1 Toman, David    http://brics.dk/~david
  2 Chomicki, Jan   http://cs.monmouth.edu/~chomick
  3 Saake, Gunter
  4 Niwinski, Damian http://zls.mimuw.edu.pl/~niwins
  ...
```

Update Deficiency

- **UPDATE** allows only attributes of the relation being updated to be present in the **SET** clauses

  $\Rightarrow$ no values from other tables can get in

- Solutions:

  1. delete followed by insert with (potentially using an auxiliary table)
  2. embedded SQL (later)
  3. extension of the **UPDATE** syntax (e.g., INGRESS):

```
UPDATE r
FROM   s1, ... , sk
SET    r.a = f(r.x,s1.y1,...,sk.yk)
WHERE  c
```


What about VIEW Updates?

- SQL's data modification commands require a base table name that is to be modified.

  $\Rightarrow$ what happens if we give a name of a **view**?

- problem: the DBMS often cannot know how a *result* of a view query was obtained from the base tables

  $\Rightarrow$ no unique way to insert/delete/modify the base tables to satisfy the modification request

Example

```
  1  create view autpub as (
  2     select name,title
  3     from   author, wrote, publication
  4     where  aid=author
  5     and    pubid=publication )

View created.

SQL> select * from autpub;

NAME           TITLE
-------------- -----------------------------------
Toman, David   Temporal Logic in Information Systems
Toman, David   Datalog with Integer Periodicity Cons
...
7 rows selected.

SQL> insert into autpub values ('foo','bar');
insert into autpub values ('foo','bar')
*
ERROR at line 1:
ORA-01779: cannot modify a column which maps to
                         a non key-preserved table
```


View Updates (cont.)

- a BIG problem:

  ⇒ views are used to provide an *external view* for the database: for a user these are the *real* tables. . .

  ⇒ at least rudimentary update capability needed

- boils down to the question id the DBMS can **uniquely** modify the database to satisfy a modification request.

  classification of views:

  ⇒ deleteble view
  ⇒ updatable view
  ⇒ insertable view
  ⇒ read-only view

- generally single-relation views (without aggregation) can be updated. The rest: depends on the DBMS. . .

Example

```
SQL> create view jlpauthor as (
  2    select *
  3    from   author
  4    where  aid in (
  5       select author
  6       from wrote, article, journal
  7       where publication=article.pubid
  8         and crossref=journal.pubid
  9    ) ) with check option;

SQL> select * from jlpauthor;

AID NAME           URL
--- -------------- ------------------------------
  1 Toman, David   http://brics.dk/~david
  2 Chomicki, Jan  http://cs.monmouth.edu/~chomicki

SQL> insert into jlpauthor values (1,'foo','bar');
1 row created.

SQL> insert into jlpauthor values (4,'baz', null);
ERROR at line 1:
ORA-01402: view WITH CHECK OPTION where-clause
                                        violation
```


Support for Transactions

- transaction starts with first **access** of the database

  ⇒ the DBMS guarantees noninterference
  (serializability) of all data access requests
  to tables in the database (using locks)

  ⇒ until it sees:

- **`COMMIT`**: make changes permanent

```
SQL> commit;

Commit complete.
```

- **`ROLLBACK`**: discard changes

```
SQL> rollback;

Rollback complete.
```

some systems insist on **`commit work`** and **`rollback work`**.

Explicit Locks

If we know we WILL access all tuples in a table (e.g., give everyone 4% salary rise), we can obtain lock on the whole table

⇒ MUCH faster execution

```
LOCK TABLE {table | view},
           ...,
           {table | view}
           IN lockmode MODE [NOWAIT]
```

⇒ **lockmode** is one of

- **ROW SHARE**
- **ROW EXCLUSIVE**
- **SHARE UPDATE**
- **SHARE**
- **SHARE ROW EXCLUSIVE**
- **EXCLUSIVE**

⇒ **NOWAIT** instructs the DBMS to return an error rather than make the application wait for the lock.

Summary

- SQL allows incremental modifications of the database
  1. insertion of tuples (**INSERT**),
  2. deletion of tuples (**DELETE**), and
  3. updating in place (**UPDATE**).

  ⇒ applies to tables and some views
- Transaction management guarantee correctness
  1. transaction starts with first data access
  2. transaction ends with

     ⇒ **COMMIT** (changes made permanent)
     transaction may be aborted by the DBMS here!

     ⇒ **ROLLBACK** (changes discarded)
- Explicit locks may improve performance (but only when you know what you're doing)