MVCC and Locks 8/5/10

PostgreSQL uses technique called Multiversion Concurrency Control (MVCC)

- db creates new copy of modified rows
- other users do not have access to modified rows until they are committed
- once committed, original rows are marked as obsolete
    - these rows remain, causing size of the table to increase until the vacuum utility is run or table is dropped and recreated
    - updating all rows in a table doubles its size!!
- Dirty Read isolation level not implemented
- see Douglas pp 164 - 168

PostgreSQL does not allow explicit choice of page-level or row-level locking

- SELECT FOR UPDATE will lock returned rows against a concurrent update

VACUUM FULL requires exclusive lock on table

documentation says that MVCC should provide better performance than locks

see Section 12.3 of PostgreSQL Documentation

MVCC

Postgres implements Multiversion Concurrency Control (MVCC) using several normally-invisible fields, notably *xmin* and *xmax*. The xmin column records the transaction id that created the row, and xmax records the transaction id that expired the row, either through an UPDATE or DELETE.

I often demonstrate MVCC by showing the xmin and xmax columns:

```
SELECT xmin, xmax FROM mytable;
 xmin | xmax
------+------
  664 |    0
(1 row)
```

Unfortunately it is hard to see a non-zero xmax column because by definition a non-zero xmax means the row is expired (or will be). I only recently realized that I can show a non-zero xmax column by deleting a row in another transaction and keeping the transaction open:

```
BEGIN WORK;
DELETE FROM mytable;
```

and then querying the table from another session:

```
SELECT xmin, xmax FROM mytable;
```

```
 xmin | xmax
------+------
  664 |  665
(1 row)
```

The *665* indicates the row will become invisible if the multi-statement transaction