Interpreting a view insertion

- We cannot insert into HappyCombo
  - After all, it is a virtual view
- However, we can use the **instead of** trigger instead
  - This turns a (patron, beer, pub) triple into three insertions of projected pairs.
  - One for Likes...
  - one for Sells...
  - one for Frequents
- For this to work, we will have to set Sells.price to null for now

HappyCombo trigger

```
create function happy_combo_func() returns trigger as $happy_combo_trig$
    begin
        insert into Likes(patron, beer) values (NEW.patron, NEW.beer);
        insert into Sells(pub, beer) values (NEW.pub, NEW.beer);
        insert into Frequents(patron, pub) values (NEW.patron, NEW.pub);
        return null;
    end;
$happy_combo_trig$ language plpgsql;
```

```
create trigger HappyComboTrig
    instead of insert on HappyCombo
    for each row
    execute procedure happy_combo_func();
```

Materialized views

- Consider:
  - For a virtual view, query is recomputed every time.
  - This can be wasteful of CPU cycles if the view is used frequently (and not modified)
- The waste of CPU cycles can be eliminated with a materialized view
- But a new problem emerges
  - Each time a base table changes...
  - ... the materialized view may change
  - However, we may still not want to pay the CPU cycles the recompute the view after each change.
- Solution:
  - Periodic reconstructions of the materialized view
  - Otherwise the view will be always out-of-date

Indexes

- (Note: text uses this for plural rather than "indices")
- Index
  - A data structure used to speed access to tuples of a relation..
  - ... given values of one or more attributes.
- Variety of index structures are possible here
  - Could be a hash table (i.e., maps a value to a tuple row)
  - In a DBMS it is almost always a balanced search tree call a B-tree
  - We will look at B-trees later...

Index definition

- Postgres:

```
create index BeerIndex on
    Beers(manf);

create index SellIndex on
    Sells(bar, beer)
```

Using indexes

- Given a value **v** for some attribute in a relation...
  - ... the index takes us to only those tuples in the relation that have **v** in the attribute of the index.
  - Without the index, we would search through the whole table (from top to bottom) for **v**.
- Example:
  - Use BeerIndex and SellIndex to find prices of beers manufactured by Phillip's and sold by The Hacked Library

Index definition

- DBMS would normally:
  - Use BeerIndex to get all beers made by Phillips.
  - Then use SellIndex to get prices of those beers, where pub = 'The Hacked Library';

```
select price from Beers, Sells
where manf = 'Phillips' and
    Beers.name = Sells.beer and
    pub = 'The Hacked Library';
```

Index used

- DBMS would normally:
  - Use BeerIndex to get all beers made by Phillips.
  - Then use SellIndex to get prices of those beers, where pub = 'The Hacked Library';

```
select price from Beers, Sells
where manf = 'Phillips' and
    Beers.name = Sells.beer and
    pub = 'The Hacked Library';
```

For our small tables, however:

- DBMS also does some other things for us under the hood (i.e., query planning)
- DBMS will decide which access method will be the least expensive for a particular query.

```
psql=> explain select price from Beers, Sells
      where manf='Phillips' and Beers.name = Sells.beer
      and pub = 'The Hacked Library';
                          QUERY PLAN
-----------------------------------------------------------
 Nested Loop  (cost=0.00..2.14 rows=1 width=4)
   Join Filter: (beers.name = sells.beer)
   ->  Seq Scan on beers  (cost=0.00..1.07 rows=1 width=84)
         Filter: (manf = 'Phillips'::bpchar)
   ->  Seq Scan on sells  (cost=0.00..1.05 rows=1 width=88)
         Filter: (pub = 'The Hacked Library'::bpchar)
```

Database tuning

- Getting a database's engineering right is tricky.
- Want to make the database run fast...
- ... and part of this means deciding what indexes to create
- Benefit:
  - An index speeds up queries that can use it.
- Drawback:
  - An index slows down all modifications on its relation because the index must be modified, too.

Tuning: thought experiment

- Suppose the only operations we performed with our Beers database were:
  - Insert new facts into a relation (10% of the time)
  - Find the price of a given beer at a given pub (90% of th time)
- Then:
  - SellIndex on Sells(pub, beer) would speed up the operations
  - However, a BeerIndex on Beers(manf) would introduce more overhead per insert (i.e. we don't use BeerIndex, yet we must maintain its accuracy).

Tuning advisors

- Because such choices on index are hard to make, there has been lots of research on **tuning advisors**
  - Hand tuning is very hard...
  - ... so some tool support would be a big help
- A tuning advisor bases its recommendations on a query load
  - Either random queries from the history of queries run on the database...
  - ... or a sample workload provided by the database's designer

Tuning advisors

- The tuning advisor then:
  - Generates candidate indexes, and..
  - ... evaluates the effect of the indexes on each workload.
- In essence:
  - Each sample query is fed to the query optimizer which assume only the proposed index is available.
  - Running time of the queries is measured...
  - ... and then examined to see if performance improves or degrades with the choice
- (Wash, rinse, repeat)

Colophon

- Some slide material is from Stanford CS145 (Jeffrey D. Ullman, Fall 2007)
- Statement-level trigger example from:
  - http://jcsites.juniata.edu/faculty/rhodes/dbms/triggers.htm