Data-Centric Programming: SQL Extensions and MapReduce

R &G - Chapter 24

Extending SQL

- **User-defined functions**
  - *Scalar* functions (UDF)
    - Operate on column values, return a value
  - *Aggregate* functions (UDA)
    - Operate on *sets* of column values, return a value
  - Table functions
    - Return a set of tuples
- **Object-Relational features**
  - Let you define *domains* in terms of OO classes

Type Extensibility

ID	Name	Salary	Zodiac
102	Bob	50K	♈

- **Tuple is an array of objects**
  - Each addressable by column name
  - Each item must belong to an appropriate "domain"
- **What domains (types) are allowed?**
  - The relational model doesn't say (or care)
  - Just has to be "atomic"
    - I.e. not other relational objects like tuples or relations

Where does this show up?

- **Predicates**
  - ```
    SELECT *
      FROM Astrologers A
     WHERE A.zodiac > 'Leo';
    ```
- **Expressions**
  - ```
    SELECT *
      FROM Astrologers A, Psychics P
     WHERE contains(A.zodiac, P.birthdate);
    ```
  - ```
    SELECT name, start_date(zodiac)
      FROM Astrologers;
    ```
- **Table Expressions**
  - ```
    SELECT *
      FROM WebCrawler();
    ```

Postgres

- **Pioneered these ideas**
  - Language support
    - Integrating objects and inheritance in the DB
    - Postgres catalogs are very interesting!
  - Efficiency
    - *Push code to data, not vice versa!*
    - Extensible Access Methods
      - And "teaching" the optimizer about them
    - Generalized Search Trees (GiST)
    - Query optimization with expensive predicates
- **PostgreSQL still one of the best**
  - Supports C, Java, Perl, Python, Ruby, R …

Examples

- **We'll focus on text, and use PL/Ruby as our language**
  - PL/Java is much faster
  - C is fastest but *dangerous!*
    - Can crash DBMS!
    - Can corrupt DB!

Consider the Classic Inverted File

doc	position	term
home/ff/cs186/ shakespeare.txt	894	In
home/ff/cs186/ shakespeare.txt	895	our
home/ff/cs186/ shakespeare.txt	896	heart's
home/ff/cs186/ shakespeare.txt	897	table;
home/ff/cs186/ shakespeare.txt	4898	heart

UDF: characters only

```
CREATE FUNCTION chars(text)
RETURNS text AS
$$
  args[0].tr('^[a-z,A-Z]', '')
$$
LANGUAGE 'plruby';
```

Getting Warmer

```
SELECT filename, pos, chars(word) as word
  FROM inverted;

SELECT filename, pos,
       lower(chars(word)) as word
  FROM inverted;

SELECT filename, pos,
       stem(lower(chars(word))) as word
  FROM inverted;
```

Getting Warmer

```
SELECT filename, pos,
  FROM inverted;

SELECT filename, pos,
       lower(chars(wo
  FROM inverted;

SELECT filename, pos,
       stem(lower(chars(word))) as word
  FROM inverted;
```

doc	position	term
home/ff/cs186/ shakespeare.txt	894	in
home/ff/cs186/ shakespeare.txt	895	our
home/ff/cs186/ shakespeare.txt	896	heart
home/ff/cs186/ shakespeare.txt	897	tabl
home/ff/cs186/ shakespeare.txt	898	heart

Term Frequency

```
CREATE VIEW termfreq AS
SELECT filename,
       stem(lower(chars(word))) as term,
       COUNT(*) as freq
  FROM inverted
 GROUP BY filename, term;


SELECT * FROM termfreq ORDER BY freq DESC;
```

User-Defined Aggregates

- **E.g. Max2: concatenate the top 2 values into a string of the form "[x, y]"**
- **State + Transition**
- **Three pieces:**
  - Initial condition of the state
  - Transition function called per new item seen
  - (Optional) final function "cooks" the state

Max2 transition function

```
CREATE FUNCTION top2(integer[2], integer)
        RETURNS integer[2] AS
$$
  tops = args[0]
  tops[0] = args[1] if args[1] > tops[0]
  if tops[0] > tops[1]
    tops[0], tops[1] = tops[1], tops[0]
  end
  return tops
$$
LANGUAGE 'plruby';
```

Max2 final and agg declaration

```
CREATE FUNCTION max2_final(integer[2])
RETURNS text AS $$
"[#{args[0][1]}, #{args[0][0]}]"
$$ language 'plruby';

CREATE AGGREGATE max2 (
  sfunc = top2,
  basetype = integer,
  stype = integer[2],
  finalfunc = max2_final,
  initcond = '{-2147483648, -2147483648}'
);
```

Table Functions

Inverting File on the Fly:

```
CREATE VIEW inverted AS
  SELECT pos, word
    FROM invert('/home/ff/cs186/
  shakespeare.small.txt') AS
      tbl(pos integer, word text);
```

Invert a file on the fly

```
CREATE FUNCTION invert(text) RETURNS setof record AS
$$
File.open(args[0], "r") do |aFile|
  pos = 0
  aFile.each_line do |line|
    line.split.each do |w|
      pos += 1
      yield [args[0], pos-1, w]
    end
  end
end
$$
language 'plruby';
```

Do extensions parallelize well?

- **UDF**

```
SELECT filename, pos, stem(lower(chars(word))) as word
  FROM table;
```

- **UDA**

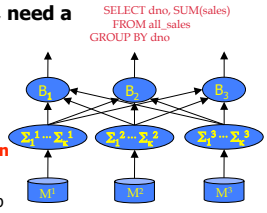
```
SELECT max2(salary)
  FROM emp;
```

- **Table Functions**

```
SELECT *
  FROM inverted;
```

Parallel Pre-Aggregation (remember?)

- **For each aggregate function, need a local/global decomposition:**
  - **sum**(S) = **Σ Σ** (s)
  - **count** = **Σ count** (s)
  - **avg**(S) = (**Σ Σ** (s)) / **Σ count** (s)
  - **UDAs need a local state function**
- **Group by:**
  - Do a local aggregate for each group
    - 3 machines (M^1..M^3), k "dno" sums each
  - Ship each local agg to an appointed "collector" to compute global agg
    - Hash function maps "dno" to one of 3 collector "buckets" (B_1..B_3)
    - Sum up each group at exactly 1 collector



Advanced topic: can you think of a summary statistic this doesn't work for?

Map/Reduce

- **Popularized by Google for parallel file munging**
- **Based on Lisp "list processing"**
- **Map function:**
  - my_map(k, v) -> list(k2, v2)
- **Reduce function:**
  - my_reduce(k, list(v)) -> list(v2)
- **Compare with UDF and UDA!**

MapReduce in Greenplum

- **Greenplum: shared-nothing PostgreSQL**
  - Also supports MapReduce
- **MapReduce scripts specified in a text file**
- **Inputs can be:**
  - (Partitioned) text files
  - Database tables
  - SQL statements
  - UNIX commands
- **Outputs**
  - Files or Database tables
- **Languages**
  - Most of the PostgreSQL extension languages
  - (But Ruby is not yet supported ☹)

Term Frequency

```
DEFINE:
  - INPUT:
      NAME:      book
      FILE:
        - localhost:/home/ff/cs186/shakespeare.txt

  - MAP:
      NAME:      wordsplit_python
      FUNCTION:  |
        for word in value.split():
          yield [word, 1]
      LANGUAGE:   python
      OPTIMIZE:   STRICT IMMUTABLE
      PARAMETERS: value text
      RETURNS:
        - key text
        - value integer
EXECUTE:
  - RUN:
      SOURCE:    book
      MAP:       wordsplit_python
      REDUCE:    SUM
```

Writing your own Reduce

```
- TRANSITION:
      NAME: summer
      LANGUAGE:   python
      PARAMETERS:
        - state integer
        - value integer
      RETURNS:
        - state integer
      FUNCTION: |
        return state + value
      LANGUAGE: python

- REDUCE:
      NAME:          my_sum
      INITIALIZE:    0
      TRANSITION:    summer
      # CONSOLIDATE: optional pre-aggregation function
      # FINALIZE:    optional state-cooker function
```

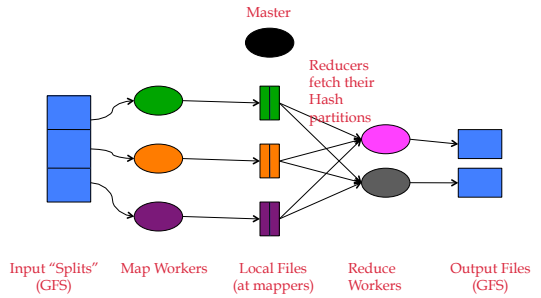
Greenplum MapReduce Arch

- **The same "shared-nothing" executor as SQL**
- **Think of PostgreSQL executor plus:**
  - Partitioned parallel scan
  - Partitioned parallel joins
  - Partitioned parallel group-by
  - Partitioned parallel sort

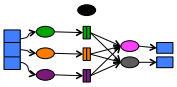
Google MapReduce (and Hadoop)

- **Language difference**
  - Reduce *guaranteed* to sort by group
  - Vs. Greenplum, which may use hash grouping
- **Architectural differences**
  - Designed for extreme scalability
    - Focus on mid-query faults and variable performance
    - *No* pipeline parallelism
  - GFS distributed filesystem for persistent I/O
    - Vs. Greenplum's DB tables or external files

Google MapReduce



Faults and Stragglers



- **Workers send "heartbeat" status reports to master**
- **If a worker doesn't heartbeat?**
  - Master starts a new one
  - Informs consumers of the new worker's output
  - What happens if old one wakes up?
- **Near the end of the job, the "stragglers" can drag you down**
  - Trick: if a worker is making slow progress, start up a clone
  - If clone wins, inform its consumers!
- **All made easy by lack of pipelining!**
  - Is this a good or bad idea?

MapReduce vs. SQL

- **A bit of a wash, but some differences:**
  - SQL has joins
    - Pretty useful for graph data
  - SQL has query optimizater
  - MapReduce can exploit input orders
    - Often clumsy to write in SQL since optimizer may mess up the order
  - Multiple output items per map or reduce
    - Vs. single item output in a UDF/UDA
    - Multiple items can be done in SQL, but clumsy
      - E.g. return a string or array, write a table function to unpack

Summary

- **Dataflow on sets is a nice programming model**
  - Can do per-row operations
    - UDF, Map
  - Can do per-group operations
    - UDA, Reduce
  - Can do table-generating operations
    - Table Functions
- **Efficient**
  - Push code to data, not vice versa
- **Auto-parallelizes!**
  - And can use simple fault-tolerance/load-balancing
- **SQL/MapReduce more similar than different**
  - As compared with other parallel programming models