Programming Assignment II

COSI 127B: Introduction to Database Systems

Eden Zik

Brandeis University

2015

Introduction

- PA1 covered skills needed to *query and modify* a database
  - Connecting to Postgres and initializing tables.
  - Inserting and deleting data.
  - Composing complex queries
- PA2 covers skills needed to *manage and integrate* a database
  - Writing integrity constraints
  - Implementing triggers
  - Using SQL with Java to power *data driven applications*

First of all, some background...

Background

- The TPC-H Benchmark
- Integirty Contraints
- Triggers
- JDBC

The TPC-H Benchmark

- A decision support benchmark for database systems.
  - Consists of a schema, data, and a standard suite of queries
  - Models the data needs of a manufacturing company
  - Used to compare performance of different database systems
- Schema
  - Eight tables populated with data.
  - Each attribute is prefixed - unique across the database

Table Name	**Prefix**	**Primary Key**
region	r	regionkey
nation	n	nationkey
supplier	s	suppkey
part	p	partkey
partsupp	ps	partkey, suppkey
customer	c	custkey
lineitem	l	orderkey, linenumber
orders	o	orderkey

Next up: Integrity Constraints

Integrity Constraints[1]

- Four types of constraints
  - Key Constraints
  - Attribute Constraints
  - Referential Integrity Constraints
  - Global Constraints
- Syntax
  - `ALTER TABLE mytable ADD PRIMARY KEY (thekey)`
  - ```
    ALTER TABLE mytable
    ADD CONSTRAINT price CHECK (price > 0)
    ```
  - ```
    ALTER TABLE mytable
    ADD CONSTRAINT price NOT NULL
    ```
  - ```
    ALTER TABLE mytable
    ADD CONSTRAINT price UNIQUE
    ```
  - ```
    ALTER TABLE mytable
    ADD CONSTRAINT fk1 FOREIGN KEY (otherkey)
    references yourtable
    ```

  When Integrity Constraints aren't enough: Triggers

[1] Covered in Lecture 6

Triggers

- Callback functions defined over tables or views
- Executed whenever a certain type of operation is performed
  - `INSERT`
  - `UPDATE`
  - `DELETE`
- Can be used to enforce sophisticated integrity constraints
  - Propagating value change
    - Ex. "When sales tax rate changes, update all prices"
    - Why not use an `UPDATE` in a loop? Establishes an internal dependency between prices and sales tax that is easier to maintain
  - Rejecting modifications leading to illegal states not otherwise enforcible by simpler constraints.
    - Ex. "A family plan cannot have more than 6 members"
    - Why not use global constraints? Too expensive

Triggers cont.

- Defined using PL/pgSQL
  - PostgreSQL specific procedural language
  - SQL with some programming facilities
    - Accepts arguments
    - Executes arbitrary SQL
    - Can return a typed value (in this case, `TRIGGER`).
  - ```
    CREATE FUNCTION some_function()
    RETURNS TRIGGER AS $$
    BEGIN
    INSERT INTO some_table(some_att) VALUES ('value');
    RETURN NEW;
    END
    $$ LANGUAGE 'plpgsql';
    ```
    ```

Triggers cont.

- Added on a table by `CREATE TRIGGER` syntax
  - Defined `BEFORE`, `AFTER`, or `INSTEAD OF` an event
  - Events can be `INSERT`, `UPDATE`, or `DELETE`'s
  - Executes `PROCEDURE` when event occurs either `FOR EACH ROW` or `FOR EACH STATEMENT`
  - ```
    CREATE TRIGGER some_trigger
    AFTER INSERT ON some_table
    FOR EACH ROW EXECUTE PROCEDURE some_function();
    ```

Finally: JDBC
    ```

JDBC

- Previously...
  - Issued queries via a Command Line Interface
  - Viewed query results as texts
- In practice...
  - Queries issued programmatically
    - JDBC = [J]ava [D]ata[b]ase [C]onnectivity
    - Enables connecting to a database via Java
    - Manipulate query results (`ResultSet`) like native data structures
  - Results of queries power applications
    - Interactivity
    - Data visualization
    - Report generation
  - We will use JDBC to visualize trends in the TPC-H database and generate reports based on part orders.

Example of a JDBC Program

```
import java.sql.*;
public class JDBCExample {
    static final String DB_TYPE = "postgresql";
    static final String DB_DRIVER = "jdbc";
    static final String DB_NAME = System.getenv("PGDATABASE");
    static final String DB_HOST = System.getenv("PGHOST");
    static final String DB_URL = String.format("%s:%s://%s/%s",DB_DRIVER, DB_TYPE,
                                                DB_HOST, DB_NAME);
    static final String DB_USER = System.getenv("PGUSER");
    static final String DB_PASSWORD = System.getenv("PGPASSWORD");
    static Connection conn;

    static final String QUERY = "SELECT 'hello world!';";

    public static void main(String[] args) throws SQLException {
        conn = DriverManager.getConnection(DB_URL,DB_USER,DB_PASSWORD);
        Statement st = conn.createStatement();
        ResultSet rs = st.executeQuery(QUERY);
        rs.next();
        System.out.println(rs.getString(1));
    }
}
```

Using JDBC

- Set database connection parameters
- Initialize a `Connection` object
- Initialize a `Statement` object using the connection
- Fill a `ResultSet` object by calling `executeQuery` on a `Statement`
- Iterate through each row in `ResultSet`
- Fetch column by number or name from the current row

So what will we do in PA2?

Overview

Part 1

- Use triggers and the integrity constraints to enforce specific semantics on the TPC-H database.
- Propagate changes in the data via the use of triggers.

Part 2

- Produce visualizations based on TPC-H data using JDBC and `gnuplot`/`graphviz`
- Generate purchase reports based on TPC-H part orders using JDBC and LaTeX.

Getting Started: Environment

- Log on to a CS Public Macine using SSH

  ```
  $ ssh [your username]@[cs public machine name].cs.brandies.edu
  ```

- Copy the provided code into your home directory

  ```
  $ cp /home/o/class/cs127b/PA2/PA2Files.tar.gz .
  ```

  ```
  $ tar -xvzf PA2Files.tar.gz
  ```

Getting Started: Database

- Set up your environment variables to connect

```
$ export PGHOST=[your database host]
$ export PGUSER=[your user name]
$ export PGDATABASE=[your user name]pa2

$ export PGPASSWORD=[your database password]
```

- Initialize the database

```
$ make initialize-db
```

- Use `psql` to test all tables have been loaded
- Reset the database as necessary

```
$ make reset-db
```

Getting Started: JDBC

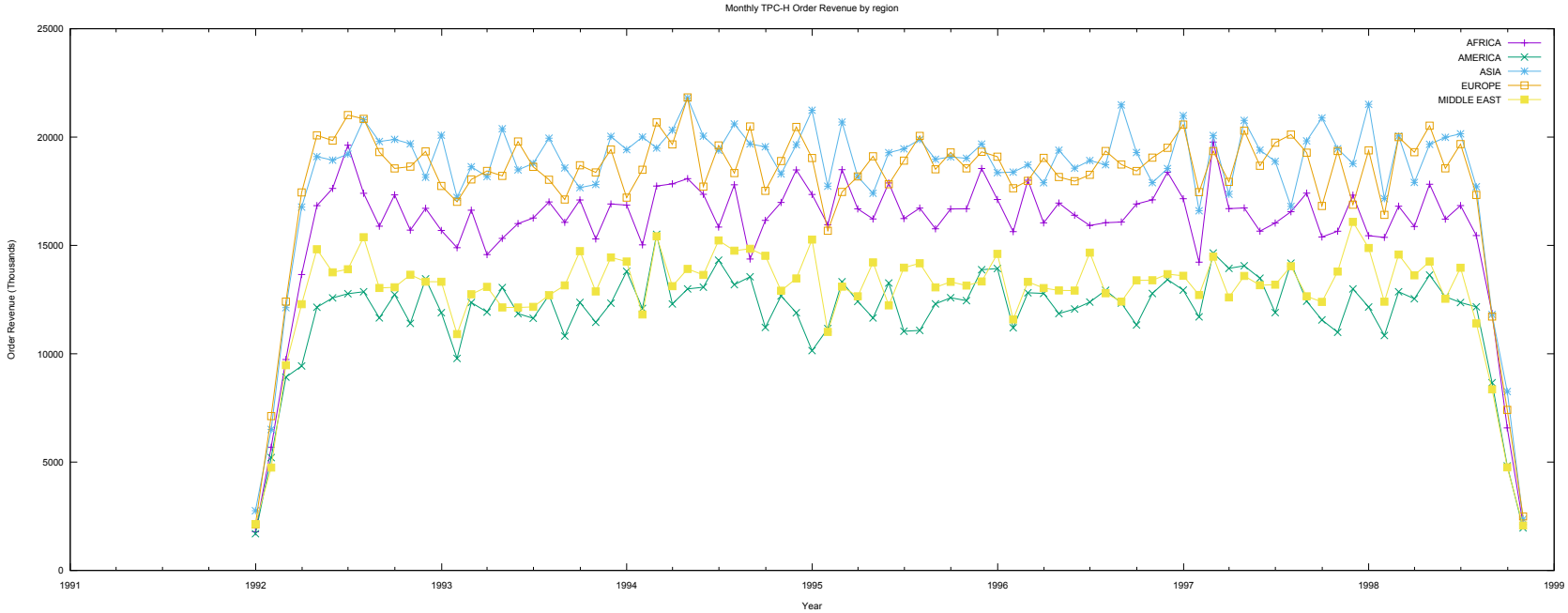
- Familiarize yourself with the provided files
  - `makefile` - A makefile to ease setup and compilation
  - `README` - this list of files
  - `lib/jdbc.jar` - The Postgres database driver
  - `lib/pa2.jar` - API's for `Gnuplot`, `Graphviz`, and `LaTex`
  - `docs/index.html` - Javadoc documentation for said API's
  - `src/part1.java` - Template code for Part 2.1
  - `src/part2.java` - Template code for Part 2.2
  - `src/part3.java` - Template code for Part 2.3
  - `txt/orders.txt` - Purchase orders for part 2.3
- Make sure your database is initialized and your environment variables set

Getting Started: Part 1.*

- Part 1.1 will require you to write primary and foreign key constraints. All of these should be saved as files under the following directory structure:
  `./sql/keys/[primary / foreign].sql`
- Part 1.2 will require you to write constraints and triggers. All of these should be saved as files under the following directory structure:
  `./sql/triggers/q_[question number].sql`

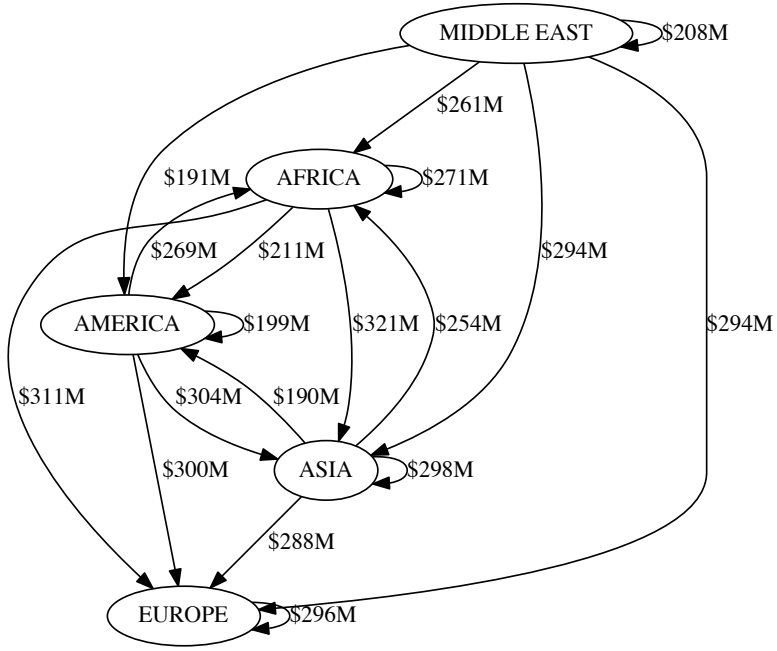
Getting Started: Part 2.1

- Part 2.1 wil require you to use an API to `Gnuplot`
  - `Gnuplot` $\rightarrow$ plotting library
  - `import edu.brandeis.cs127b.pa2.gnuplot.*;`
  - Documented in `docs/*`
  - Starter code produces a simple plot ($sin(x)$)
  - Your task: *calculate supplier sales totals by region*



Getting Started: Part 2.2

- Part 2.2 will require you to use an API to `Graphviz`
  - `Graphviz` $\rightarrow$ graph library
  - `import edu.brandeis.cs127b.pa2.graphviz.*;`
  - Documented in `docs/*`
  - Starter code produces a simple graph (random edges)
  - Your task: *calculate the total sales between suppliers and customers grouped by the regions where the suppliers and customers are located*



Getting Started: Part 2.3

- Part 2.3 will require you to use an API to LaTeX
  - LaTeX→ document typesetting library
  - `import edu.brandeis.cs127b.pa2.latex.*;`
  - Documented in `docs/*`
  - Starter code produces a simple document (empty order)
  - Your task: *determines minimum prices for each of a specified set of items in a file, and generate a purchase order that orders each item from its cheapest supplier*

Brandeis University

415 South St. (781) 736-2000
Waltham, MA 02453

Date:
10/24/2015
Order #:
4

Part #	Quantity	Unit Price	Amount
Supplier: 106			
1223	95	$253.55	$24087.25
Subtotal			**$24087.25**
Supplier: 108			
4801	37	$344.32	$12739.84
Subtotal			**$12739.84**
Supplier: 109			
1822	50	$53.60	$2680.00
3098	12	$116.39	$1396.68
3229	59	$392.26	$23143.34
4714	4	$14.38	$57.52
Subtotal			**$27277.54**
Supplier: 11			
1206	15	$82.20	$1233.00
679	4	$177.54	$710.16
Subtotal			**$1943.16**
Total			**$66047.79**
# of Suppliers			**4**

Getting Started: Part 2.*

- Compile your code in `src/*` as follows:
  `$make build-[part1 / part2 / part3]` → `./bin`
- Run your code as follows:
  `$make run-[part1 / part2 / part3]` →
  `./[plot/gv/tex]`
- Visualize your result as follows:
  `$make vis-[part1 / part2 / part3]` → `./vis`
- Submit a well commented source code in `./src`, along with your `gnuplot`, `graphviz`, and LaTeXcode and their resulting visualizations
- A detailed writeup about your algorithm for each part in a PDF format in `./pdf`.

Submission

1. Set the proper environment variables:

   ```
   $ export FIRSTNAME=[your first name as it appears on LATTE]
   $ export LASTNAME=[your last name as it appears on LATTE]
   ```

2. Type the following command to package all relevant files into a submittable file:

   ```
   $ make submit
   ```

   A file with the name `[your last name]_[your first name]_127b_pa2.tar.gz` will be created in the current directory.

3. Use secure copy to transfer the file from the server to your machine:

   ```
   $ scp [your username]@[cs public machine name].cs.brandies.edu: /[pa2]/*_*_127b_pa2.tar.gz .
   ```

   The file should then be transferred to the current directory on your local machine

4. Upload the file to LATTE.