MPCS 53001: Databases

Zach Freeman

Lecture #4

Class announcements

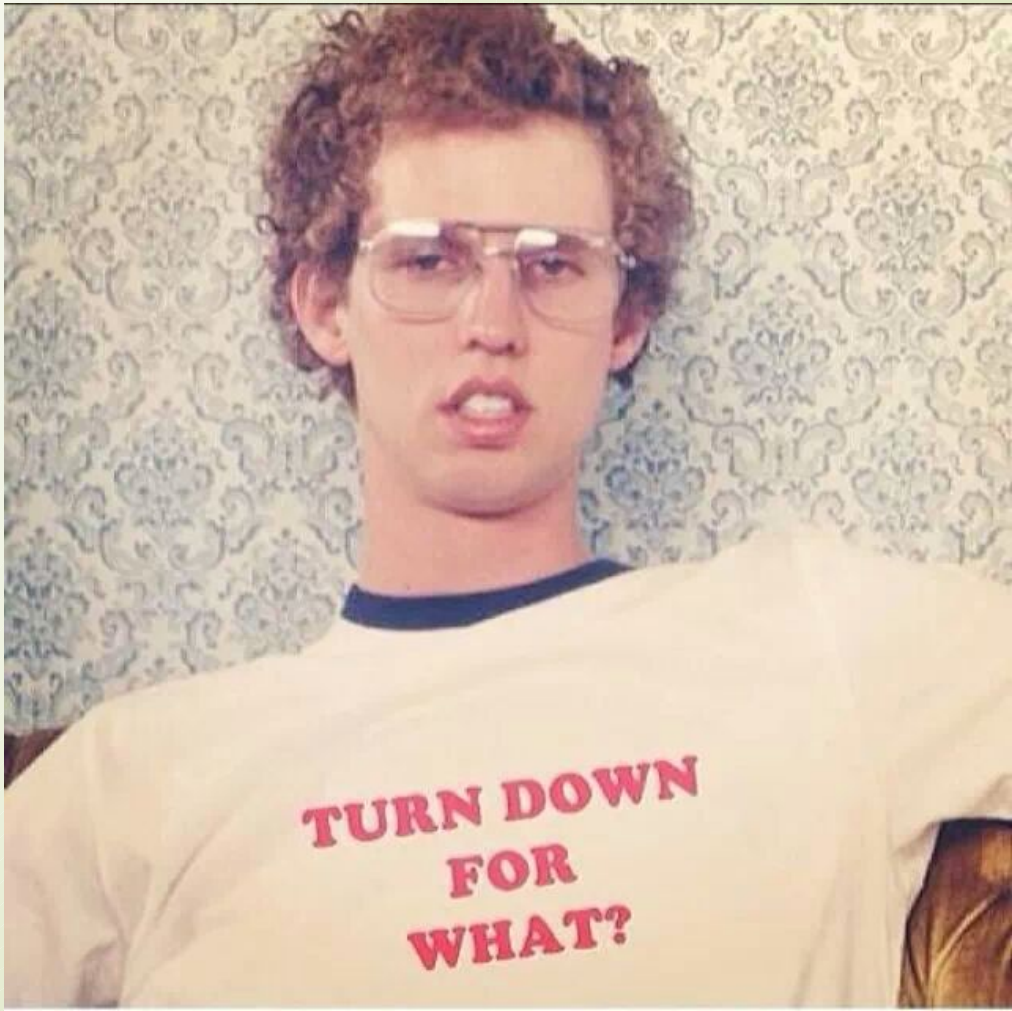
- Midterm in two weeks (10/31 & 11/2).
  - Open book/notes (no devices)
  - Next week: practice take home test
- Assignment 4:
  - Gradiance problem set
  - Final Project – multi-relation queries and aggregation

Recap

- From Design to Relations to Schema
- Normalization
- Decomposition
- Relational Algebra
- SQL…
- Today: MORE SQL.

Overview for today

- *Multirelation* SQL queries
- Comments
- Subqueries
- Aggregation
- Modifications
  - Data
  - Schema



Running Example Tables

Race(raceName, distance, raceCountry, registrationCap)

Runner(runnerName, age, yearsRunning, favRace)

AgeGroup(ageGroupCode, ageGroupDesc, ageGroupMin, ageGroupMax)

Likes(runnerName, raceName)

Registrations(runnerName, raceName)

Results(raceName, runnerName, place, ageGroup)

SQL Comments

- Comments in SQL:
  - Provide context for complex code
  - Help explain what code does

```
-- this is a comment
-- this query gets all the runners
SELECT *
FROM Runner;
```

Single Relation Queries

- Only operate on one table
- Single tables contain very specific information
  - By design (due to normalization)
- Usually we want related information from several tables

Single Relation Queries

- Example:
- Find all races that have “Marathon” in the name and are not held in the USA.

Multirelation Queries

- Can operate on any number of tables
- List multiple tables in the FROM clause
  - Conditions in the WHERE clause
- Allows us to pull more complex data sets

Multirelation Queries

- Example:
- Find the name, age and years running of all runners who have won races.

Multirelation Queries

- What if different tables have columns with the same names?
- Relation-**dot**-attribute notation:
  - <table name>.<column name> disambiguates attributes with the same name from several relations
- Example: find all races that runners registered for the Boston Marathon like.

Aliases

- Sometimes we need to refer to two or more copies of the same relation (table).
- You can use aliases for the tables.
- Example: Find all pairs of different races liked by the same runner.

Likes(runnerName, raceName)

Formal Semantics

1. Start with the product of all relations in the FROM clause.
2. Apply σ (selection), using condition(s) in WHERE clause.
3. Apply π (projection) using attributes in the SELECT clause.

- Essentially, the same thing as single-relation query with the addition of product (X) of all relations.

FOR loop comparison

- Imagine a variable representing a row in each table in the FROM clause.
- Imagine a series of nested loops over these variables that produces every possible combination of rows, one from each of the relations in the FROM clause.
  1. For each row combination, check if it satisfies the WHERE clause.
  2. If so, print the values of terms in the SELECT clause.

In-class exercise

- Find all runners that are registered for their favorite race.
- Find all runners that finished two different races in the same place.
- Find all runners that finished three different races in the same place.

Subqueries

- Subquery: the result of a query that can be used in the WHERE clause of another query.
- Simplest case: subquery returns a single unary tuple (a single atomic value)

Example

- Find the runner that finished the NYC Marathon in the same place as Rita Jeptoo finished the Chicago Marathon.

```
SELECT runnerName
FROM Results
WHERE raceName = 'NYC Marathon‘
AND place = (SELECT place
        FROM Results
        WHERE runnerName = 'Rita Jeptoo‘
        AND raceName = 'Chicago Marathon')
```

subquery

Scoping rule

- Attribute refers to the most closely nested relation with that attribute.
- Parenthesis around subquery are essential.

The IN operator

- Previous example – looking for SINGLE value. What about multiple values?
- IN returns true if the tuple is IN the relation.
- Find the name and registration cap of all races that Meb likes.

The EXISTS operator

- EXISTS(relation)
- Returns true if the relation contains at least one tuple (nonempty)

Unique race country

Find the races in the Race table that are the unique race in their country.

```
SELECT raceName
FROM Race r
WHERE NOT EXISTS (SELECT *
                  FROM Race
                  WHERE raceCountry =
r.raceCountry                    AND raceName !=
r.raceName)
```

Correlated subquery

- A subquery that refers to values of a surrounding query is called a *correlated subquery*.
- A correlated subquery must be evaluated (by the DBMS) for every tuple in the outer query.

Correlated subquery example

- Find all runners registered for their favorite race

```
SELECT *
FROM Registrations r
WHERE runnerName IN
    (SELECT runnerName    FROM Runner
                WHERE favRace = r.raceName);
```

Quantifiers

- ANY is an existential quantifier

  attribute condition ANY (subquery)
- ***At least one row*** in the result of the subquery satisfies the condition

- ALL is a universal quantifier

  attribute condition ALL (subquery)
- ***All rows*** in the result of the subquery satisfy the condition.

Quantifier Example 1

- Find the race(s) with the highest registration cap.

Quantifier Example 2

- Find the races in the USA that have neither the largest registration cap nor the lowest (of all races in the USA).

Set Operators

- UNION, INTERSECT
- (subquery) UNION (subquery)
- Only UNION is supported in MySQL but you can write equivalent queries for INTERSECT.

UNION Example

- Find the runners that have won marathons or that are registered for the Chicago Marathon.

```
(SELECT    runnerName
FROM       Results
WHERE      place = 1
AND        raceName LIKE '%marathon%')
UNION
(SELECT    runnerName
FROM       Registrations
WHERE      raceName = 'Chicago Marathon');
```

INTERSECT Example

- Find the runners that have won marathons and that are registered for the Chicago Marathon.

```
(SELECT    runnerName
FROM       Results
WHERE      place = 1
AND        raceName LIKE '%marathon%‘)
INTERSECT
(SELECT    runnerName
FROM       Registrations
WHERE      raceName = 'Chicago Marathon');
```

Forcing Set/Bag Semantics

- Default for select-from-where queries is bag; default for union is set
  - Bag is chosen for performance
- Force set semantics (no duplicates) with DISTINCT after SELECT
- Force bag semantics with ALL after UNION

DISTINCT Example (reminder)

- Find all different distances of races

Aggregation

- Format: Aggregate-function(attribute)
- SUM, AVG, MIN, MAX, COUNT
- Use these in SELECT clause

Aggregation (briefly explained)

- SUM – summation
- AVG – takes average
- MIN – finds minimum
- MAX – finds maximum
- COUNT – only counts non-NULL values
- COUNT(*) – counts ALL rows

Average Place for Tera Moody

Find the average finishing place for Tera Moody

Eliminating duplicates before aggregation

- Find the number of different places that Rita Jeptoo has finished in.

- DISTINCT can be used with aggregation.

Lowest place and race for Dean Karnazes

- First attempt (very common mistake):

```
SELECT raceName, MIN(place)
FROM Results
WHERE runnerName = 'Dean Karnazes';
```

- Illegal query in SQL!*
- How do we find the race?

*Executed in previous version of MySQL but gave incorrect result. No longer executes. Good job MySQL!

MS SQL will tell you: <Attribute> is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause.

Grouping

- GROUP BY
- Aggregation applied to several subsets of your relation grouped together by some condition
  - So far the entire relation has been aggregated
- GROUPs BY a list of attributes

Grouping Example 1

- Find the MAX (worst) place for each runner.

Grouping explanation

- The relation that is the result of the FROM and WHERE clauses is grouped according to the values of the attributes of the GROUP BY clause and aggregations take place within each group.

- One result tuple is produced from each group.

Previous Example Corrected

- Find the lowest place (and which race it was in) for Dean Karnazes.

Grouping Example 2

For each runner, find the average registrationCap of the races for which they’re registered.

Restrictions on SELECT lists with aggregation

- If aggregation with grouping is used, then each element of the SELECT clause must either:

1. Be aggregated
2. Appear in the GROUP BY clause

HAVING clause

- Selections on groups just as WHERE clauses are selections on tuples.
- Conditions can use tuple variables or any attributes in the relations in the FROM clause.
  - Tuple variables range only over the group
  - Attributes must be grouping attributes or be aggregated

HAVING example

- Find the average place of runners that have either finished at least 3 races or have been running for more than 20 years.

HAVING vs WHERE

- All conditions we learned about for WHERE clause will work for HAVING (including wildcard)
- Difference between HAVING and WHERE:
  - WHERE filters before data is grouped
  - HAVING filters after data is grouped
- The difference is important because rows eliminated by the WHERE clause will not be included in the group

Having vs WHERE

```
SELECT column_name,
aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name
HAVING aggregate_function(column_name) operator
value;
```

-from W3Schools.com

- Find all runners that have finished three or more races in the top 10.

Results(runnerName, raceName, place, ageGroup)

More HAVING examples

- Find all runners registered for more than 2 races.
- Find all runners who are over 35 and are registered for more than 2 races.

Relations:

Registrations(raceName, runnerName)

Runner(runnerName, age, yearsRunning, favRace)

Database modifications

- Change the current relation instance
- Results of modifications last beyond your current session
- Insert a new tuple
- Delete a current tuple
- Update a current tuple
  - Change the value of one or more of its attributes

Insert

INSERT INTO relation

VALUES(list-of-values)

Insert the tuple defined by the list of values, associating values with attributes in the order the attributes were declared.

You can also list the attributes as arguments of the relation.

INSERT INTO Likes(raceName, runnerName)

VALUES ('Big Sur International Marathon','Deena Kastor');

Insert the result of a query

INSERT INTO relation

(subquery)

- Example: Find potential running buddies for Zach by selecting all runners who like the races that Zach likes.

```
CREATE TABLE PotentialRunningBuddies (
    name VARCHAR(50),
    raceName VARCHAR(50)
)
```

Potential buddies

```
INSERT INTO PotentialRunningBuddies
(SELECT DISTINCT B.runnerName,
B.raceName FROM Likes A, Likes B
WHERE A.runnerName = 'Zach Freeman'
AND    B.runnerName != 'Zach Freeman'
AND    A.raceName = B.raceName);
```

DELETE

DELETE FROM relation

WHERE condition

Delete all tuples from the relation satisfying the condition. **If no condition is given all tuples are deleted** (usually DROP/CREATE TABLE is better).

```
DELETE FROM Likes
WHERE   runnerName = 'Zach Freeman'
AND   raceName = 'Hot Chocolate'
```

UPDATE

UPDATE relation
SET list-of-assignments
WHERE condition

Example: Make runners younger

```
UPDATE  Runner
SET    age = 22
WHERE   age > 22;
```

Schema Modifications

- Never* modify the schema of a live database
- Schema modification may result in:
  - Disk fragmentation
  - Rebuilding indexes
  - Re-optimizing queries
  - Run time errors of existing SQL queries

*Do it suuuuper carefully.

Changing Columns - ADD

Adding columns (low risk)

ALTER TABLE <table_name>

ADD <column_name> <data_type>

- Add an attribute to an existing relation

ALTER TABLE Runner

ADD shoeSize VARCHAR(10) DEFAULT 'unknown';

Changing Columns - ALTER

Altering columns (some risk)

ALTER TABLE <table_name>

MODIFY COLUMN <column-name> <datatype>

ALTER TABLE Runner

MODIFY COLUMN runnerName varchar(100);

ALTER TABLE <table_name>

CHANGE <oldColumn> <newColumn> <datatype>

ALTER TABLE Runner

CHANGE favRace favoriteRace varchar(50);

Changing Columns - DROP

Dropping columns (very risky)

ALTER TABLE <table_name>

DROP <columnName>

ALTER TABLE Debate

DROP mic

Summary

- Multirelation SQL queries (JOINS next week!)
- Subqueries
- Aggregation (MAX, MIN, AVG, COUNT, SUM)
- Modifications
  - Data
  - Schema

- HW4 – keep up the good Piazza usage.

Next Week

- Joins
- More Subqueries and Aggregation (w/JOINs)
- Temporary Tables
- Views
- Indexing
- Constraints
- Practice Take-Home Midterm

Homework 4

- Multirelation queries
- Aggregation queries
- Modifications

Be sure to include CREATE/POPULATE scripts with your homework 4 submission!


QUESTIONS?
COMMENTS?
CONCERNS?