MPCS 53001: Databases

Zach Freeman

Lecture #5

Class announcements

- Midterm next week (10/31 & 11/2)
  - Open book/notes
  - Practice take home test distributed at end of today's class
- Assignment 5 due at normal time (5:30 before class).
  - 5th Gradiance problem set (SQL lab)
  - 5th part of final project
    - JOINS (derived tables)
    - Aggregation
    - Views

Recap

- CREATE/POPULATE methods
- Comments
- Single Relation SQL queries
  - SELECT, FROM, WHERE
- Multirelation SQL queries
  - Subqueries
- Aggregation
  - GROUP BY/HAVING
- Modifications (Data & Schema)

Overview for today

- Joins
  - (New syntax for multi-relation queries)
- More Subqueries and Aggregation (using JOINs)
  - Derived tables
- Temporary Tables
- Views
- Indexing
- Constraints

Joins

- Any multi-relation SQL query (what we looked at last week) is a join
- Last week JOIN condition(s) were described explicitly in the WHERE clause
- New format using JOIN. General form:

relation1 [join_type] JOIN relation2

ON [join_condition(s)]

JOIN basics

- Compare:
  - JOIN = ,
  - ON = WHERE

JOIN Example 1

- For each runner, find the distance of his/her favorite race.

```
SELECT r1.runnerName, r2.distance
FROM Runner r1, Race r2
WHERE r1.favRace = r2.raceName;
```

```
SELECT r1.runnerName, r2.distance
FROM Runner r1 JOIN Race r2
ON r1.favRace = r2.raceName
```

JOIN Example 2

- Find pairs of runners who like the same race.

Last week:

```
SELECT A.runnerName, B.runnerName,
A.raceName
FROM Likes A, Likes B
WHERE A.raceName = B.raceName
AND A.runnerName > B.runnerName;
```

INNER JOIN Example

- Find all info on runners and the races they are registered for.

```
SELECT *
FROM Runner run
INNER JOIN Registrations reg
    ON run.runnerName = reg.runnerName
INNER JOIN Race r
    ON reg.raceName = r.raceName
```

INNER JOIN Example 2

- Find the names and distances of all races runners have finished as well as the runner's current age and age group for that finish.

```
SELECT run.runnerName, run.age, r.raceName,
r.distance, ag.ageGroupDesc
FROM Runner run
INNER JOIN Results res
    ON run.runnerName = res.runnerName
INNER JOIN Race r
    ON res.raceName = r.raceName
INNER JOIN AgeGroup ag
    ON res.ageGroup = ag.ageGroupCode
```

JOINs with Using

Relation1 JOIN Relation2 USING (attributes)

- Equate the attribute(s) of the two relations stated in the USING clause and keep a single copy.
  - Both relation1 and relation2 should have the stated attribute(s)

JOIN example with Using

Find pairs of runners liking the same race, along with the name of the race.

Natural Joins

relation1 NATURAL JOIN relation2

- The join conditions are implicit so:
  - ON or USING is not allowed (or needed).
  - WHERE is still allowed.
- Attributes with the same name are also required to have the same value type.
- A single attribute copy is kept in the result (same as USING)

Natural Join Example 1

- For each runner, find the races that the runner is registered for and also likes.

```
SELECT runnerName, raceName
FROM Likes
NATURAL JOIN Registrations;
```

Natural Join example 1

- For each runner, find the races that the runner is registered for and also likes.

How would we write the same query using INNER JOIN?

Natural Join example 2

- Find the results of all runners that finished a race that they like, along with the distance of that race.

Natural Joins and Duplicates

- Duplicate attributes (columns) are eliminated but duplicate rows are not.

```
SELECT *
FROM Likes
NATURAL JOIN Race
```

OUTER Join

- Join relation R and relation S on condition(s) C.
- Tuples of R that do not join with any tuple of S are called *dangling tuples*.
- Similar to INNER JOIN, **but** OUTER JOIN keeps dangling tuples in the result, padded with NULLs.

Outer Join Example

- List all runners and, if they have any, their liked races.
- Every tuple of Runner will be represented in the result at least once; some will have NULL as likedRace.

```
SELECT r.runnerName AS runner, l.raceName AS likedRace
FROM Runner r LEFT OUTER JOIN Likes l
ON r.runnerName = l.runnerName;
```

OUTER JOIN Example 2

- List all runners without a liked race.

Outer Join Varieties

FULL, LEFT, RIGHT OUTER JOIN

- FULL: Every tuple of the relation on the left and right of the join will be represented in the result, padded with NULLs if necessary.
  - No FULL OUTER JOIN in MySQL ▮▮
- LEFT/RIGHT: NULL padding for attributes of the indicated relation in the SELECT clause.

JOIN type reminder

- INNER (default)
- OUTER
- USING
- NATURAL

RECAP: Subqueries in the WHERE clause

- Subqueries (including subqueries performing aggregation) can be used in the WHERE clause.

- **EXAMPLE:** Find all runners (along with the race name and their finishing place) who have a finish in a better (lower) position than Ryan Hall’s best finish.

NEW: Subqueries in the FROM clause

- Subqueries can also be used in the FROM clause and referenced as tables.
- These are called **derived tables**.

- **EXAMPLE:** Find all runners that place, on average, better (lower) than the average place of Kara Goucher.

Subqueries in the FROM clause w/JOIN

- When referenced in the FROM clause, subqueries can be joined to like any other table.

- **EXAMPLE:** Find all races that are a longer distance than the average distance of all races in that race's country.

Subquery in FROM and HAVING example

- Find all runners that have finished **all** races that Kara Goucher has finished.

Temporary Tables

- Format: CREATE TEMPORARY TABLE
- Not permanent (surprise!). Lasts til session closes.
- Useful for:
  - Replacing frequently used subqueries.
  - Simplifying a complex query.
  - Creating tables that don’t exist but that you need (temporarily).

Temporary Tables (continued)

- Table is automatically dropped when session ends.
- The same temporary table can be created in multiple sessions without affecting each other.

Temporary Table example

- Find all runners that have finished **all** races that Kara Goucher has finished.

```
CREATE TEMPORARY TABLE KaraRaces AS
SELECT raceName
FROM Results
WHERE runnerName = 'Kara Goucher';
```

Views

- A view is a relation defined as the result of a query over existing relations (called base tables). Two types of views:

1. Virtual – only expressed as a query
2. Materialized – stored in a table

Declaring Views

CREATE VIEW <view_name>(attributes)

AS <query>

- The list of attributes is optional; if not specified, the attribute names from the query are used.
- Views speed up/simplify query writing (but do NOT speed up processing).

View Example

- For every result, show the name and age of the runner.

```
CREATE VIEW AgeResults AS
SELECT r.raceName, r.place, ag.ageGroupDesc,
r.runnerName, run.age AS currentAge
FROM Results r
INNER JOIN Runner run
    ON r.runnerName = run.runnerName
INNER JOIN AgeGroup ag
    ON r.ageGroup = ag.ageGroupCode;
```

Querying Views

- Views can be accessed (SELECTed FROM) like ordinary relations.
- What is Dean Karnazes' current age and what age group was he in when he ran the Badwater Ultramarathon?

Processing View Queries

The view query is translated into a query over the base relations.

SELECT currentAge, ageGroupDesc

FROM (***SELECT r.raceName, r.place, ag.ageGroupDesc,***

r.runnerName, run.age AS currentAge

FROM Results r

INNER JOIN Runner run ON r.runnerName = run.runnerName

INNER JOIN AgeGroup ag ON r.ageGroup = ag.ageGroupCode) AgeResults

WHERE runnerName = 'Dean Karnazes‘

AND raceName = 'Badwater Ultramarathon';

Indexes

- Data structures used to speed access to tuples of a relation given the values of some attribute(s)
- Speed up queries by using binary search instead of linear search

Indexing Motivation

- For big relations, often only a small fraction of tuples satisfy the WHERE clause of a query.
- In the absence of indexing, every tuple must be examined to produce the result.

```
SELECT runnerName
FROM Runner
WHERE favRace = 'Chicago Marathon'
```

Declaring Indexes

CREATE INDEX index_name

ON relation(attributes)

- No restriction on which attributes (do not have to be unique)

CREATE INDEX FavRaceIndex
ON Runner(favRace)

Implicit Indexing

- MySQL creates a unique index on:
  - The primary key of any relation
  - Any unique attribute(s)

```
CREATE TABLE AgeGroup(
ageGroupCode char(4),
ageGroupDesc varchar(15) UNIQUE,
ageGroupMin int,
ageGroupMax int,
PRIMARY KEY(ageGroupCode));
```

Using Indexes

- Given an indexed attribute and a condition on it, the index lets you directly select only the tuples that satisfy the condition.

```
SELECT runnerName
FROM Runner
WHERE favRace= 'Chicago Marathon'
```

Using Indexes Example

```
SELECT raceName, r.runnerName,
r.age
FROM Likes l
INNER JOIN Runner r
    ON l.runnerName = r.runnerName
WHERE favRace = 'Chicago Marathon';
```

Indexes and Joins

- Indexes may or may not be useful in computing joins.
- Usefulness depends on relation statistics.

```
SELECT runnerName, raceName
FROM Likes
NATURAL JOIN Results
NATURAL JOIN Registrations
```

Indexes Pros and Cons

- Speed up some single relation queries
- May speed up some joins

- Slow down modifications
- Take up additional storage space on disk and in memory

- Indexes must be chosen for a specific query mix.

Index Tips

- Primary keys/UNIQUE attributes are already indexed – no need to add additional indexes on those fields.
- Large string attributes are bad choices for indexes.
- Check in on indexes from time to time to fine tune.

Index Guide

- MENTOR* your indexes
- M – Measure
- E – Explain
- N – Nominate
- T – Test
- O – Optimize
- R – Rebuild

*Bill Karwin, “SQL Antipatterns”

Constraints

- Restrictions on the data in your database.
- E/R diagrams support a limited number of restrictions: keys, one-one, many-one, relationships
- DBMS allow much more fine-tuning of constraints.
- Ideally, all constraints are enforced in the database.

Constraint Types

- Primary key declaration
- Foreign keys (referential integrity)
- Attribute- and tuple-based checks
  - Within a relation
- SQL assertions (global constraints)
- Triggers (future class topic)

Primary Keys

- Every table needs a primary key.
- Allows identification of a unique row.
- Use the primary key that works for a given table (can vary throughout your database)

Foreign Keys

- Basically: Relates a field in one table to a field in another.
- More specifically: In relation X a clause that says attribute A references attribute B in relation Y so whatever values appear in the A column of X must also appear in the B column of Y.
  - Usually in relations corresponding to E/R relationships.
- Attribute B must be declared the PRIMARY KEY or at least UNIQUE in Y.

Foreign Key Example

CREATE TABLE Runner(

runnerName varchar(50) PRIMARY KEY,

age int,

yearsRunning int,

favRace varchar(50));

CREATE TABLE Likes(

raceName varchar(50),

runnerName varchar(50),

FOREIGN KEY(runnerName) REFERENCES Runner(runnerName),

PRIMARY KEY(raceName, runnerName));

SQL Server Declaration

```
CREATE TABLE Likes(
    raceName varchar(50),
    runnerName varchar(50) REFERENCES Runner(runnerName),
    PRIMARY KEY(raceName, runnerName));
```

Foreign Keys in MySQL

- Both the referenced and the referencing tables must be using storage engine **InnoDB** (our default, luckily).
  - FOREIGN KEY syntax must be used
- In the referenced table there *__must__* be an index on the referenced columns
  - PRIMARY KEY or UNIQUE creates one automatically.

MySQL Foreign Key Example

```
CREATE TABLE Runner(
runnerName varchar(50) PRIMARY KEY,
age int,
yearsRunning int,
favRace varchar(50) ENGINE = InnoDB;

CREATE TABLE Likes(
raceName varchar(50),
runnerName varchar(50),
FOREIGN KEY(runnerName) REFERENCES Runner(runnerName),
PRIMARY KEY(raceName, runnerName) ENGINE = InnoDB;
```

Foreign Key Constraint Violations

- Inserting or updating a Likes tuple so that it refers to a nonexistent runner is always rejected.
- Deleting or updating a tuple in Runner that has a runnerName value that some tuple in Likes refers to offers several options:

1. **Default**: reject the modification.
2. **Cascade**: propagate changes to referring Likes tuples.
3. **Set Null**: change referring tuples to have NULL in referring attributes.

Cascade Example

CREATE TABLE Likes(

raceName varchar(50),

runnerName varchar(50),

FOREIGN KEY(runnerName) REFERENCES Runner(runnerName),

ON UPDATE CASCADE ON DELETE CASCADE,

PRIMARY KEY(raceName, runnerName));

- What happens in Likes when we update “Rita Jeptoo” to “Cheata Jeptoo” in Runner?

- What if we delete Rita Jeptoo (or Cheata Jeptoo) from Runner?

Set NULL Example

- Add Foreign Key on Results(ageGroup) referring to AgeGroup.
- What if we update ageGroupCode from 3039 to 3040 in AgeGroup?
- Delete age group 50-59?

Selecting a Policy

- Add ON [DELETE|UPDATE]

[CASCADE, SET NULL] to foreign key declaration

FOREIGN KEY(runnerName)
REFERENCES Runner(runnerName)

ON DELETE SET NULL

ON UPDATE CASCADE

“Correct” policy is a design decision.

Attribute-Based Checks

- Follow an attribute by a condition that must hold for attribute values.
- CHECK(condition)
  - Condition may involve the checked attribute
  - Other attributes or relations may be involved but only in a subquery.
- The CHECK clause is parsed but ignored by all storage engines in MySQL ⏸

Tuple-Based Checks

- Separate element of table declaration.

CHECK(condition)

- Condition can refer to any attribute or relation.
- Checked whenever a tuple is inserted or updated.
- MySQL parses but ignores the check.

Guaranteed Race Size Restriction

- Only NYC Marathon can have results higher than 50000.

TABLE Results(

raceName varchar(50),

runnerName varchar(50),

place int,

ageGroup char(4),

PRIMARY KEY(raceName, runnerName),

CHECK (raceName = 'New York City Marathon' OR place <=50000));

SQL Assertions

- Database schema constraints.
- Checked whenever an involved relation changes.
- “A Boolean-valued SQL expression that must be true at all times.”

CREATE ASSERTION <assertion_name>

CHECK (<condition>)

- Not supported in MySQL but triggers can be used to achieve similar effect.

Assertion Example

- There must be at least twice as many runners as there are races (business rule)

```
CREATE ASSERTION RaceLimit
CHECK (
    (SELECT COUNT(*) FROM Race) <=
    (SELECT COUNT(*)/2 FROM Runner)
    );
```

Summary

- Joins
- Subqueries and Aggregation
- Temp Tables
- Views
- Indexing
- Constraints
  - Primary Keys, Foreign Keys, Checks, Assertions
- Midterm next week!
  - Sample Midterm today!

Midterm Review

- Practice take-home exam!
- ER diagrams
- ERD to relations
- Functional Dependencies, normalization, normal forms
- SQL
  - Joins (INNER, OUTER, USING, NATURAL)
  - Aggregation (GROUP BY, HAVING)
  - Subqueries (including derived tables)
  - Views
  - Temp Tables

Homework 5

- Views/Temp Tables
- JOINs
- Aggregation

Be sure to include CREATE/POPULATE scripts with your homework 5 submission!


QUESTIONS?
COMMENTS?
CONCERNS?

Practice aggregation queries

- Find the runner who likes the most races.
- Find the distance of the race(s) with the most registered runners.