CMSC424: Database Design

SQL

February 17, 2020

Instructor: Amol Deshpande
amol@cs.umd.edu
Today’s Plan

- SQL (Chapter 3, 4) – Remaining Stuff
  - Triggers (5.3)
  - Authorization (4.6), Ranking (5.5)
  - Some Complex SQL Examples

- Project 1 discussion on Wednesday

- Entity-Relationship Modeling

- Wednesday: Anatomy of a Web Application
  - Project 2
Triggers

- A **trigger** is a statement that is executed automatically by the system as a side effect of a modification to the database.

- Suppose that instead of allowing negative account balances, the bank deals with overdrafts by
  
  1. setting the account balance to zero
  2. creating a loan in the amount of the overdraft
  3. giving this loan a loan number identical to the account number of the overdrawn account
create trigger overdraft-trigger after update on account
referencing new row as nrow
for each row
when nrow.balance < 0
begin atomic
    actions to be taken
end
create trigger overdraft-trigger after update on account
referencing new row as nrow
for each row
when nrow.balance < 0
begin atomic
    insert into borrower
    (select customer-name, account-number
    from depositor
    where nrow.account-number = depositor.account-number);
    insert into loan values
    (nrow.account-number, nrow.branch-name, nrow.balance);
    update account set balance = 0
    where account.account-number = nrow.account-number
end
Triggers...

- External World Actions
  - How does the DB order something if the inventory is low?

- Syntax
  - Every system has its own syntax

- Careful with triggers
  - Cascading triggers, Infinite Sequences...

- More Info/Examples:
  - Google: “create trigger” oracle download-uk
Example: find which courses are a prerequisite, whether directly or indirectly, for a specific course

```
with recursive rec_prereq(course_id, prereq_id) as (  
  select course_id, prereq_id  
  from prereq  
  union  
  select rec_prereq.course_id, prereq.prereq_id,  
  from rec_rereq, prereq  
  where rec_prereq.prereq_id = prereq.course_id
)
  select *  
  from rec_prereq;
```

Makes SQL Turing Complete (i.e., you can write any program in SQL)

But: Just because you can, doesn’t mean you should
Ranking is done in conjunction with an order by specification.

Consider: \( student\_grades(ID, GPA) \)

Find the rank of each student.

\[
\text{select } ID, \text{ rank() over (order by } GPA \text{ desc)} \text{ as s\_rank} \\
\text{from } student\_grades \\
\text{order by s\_rank}
\]

Equivalent to:

\[
\text{select } ID, (1 + (\text{select count(*)} \\
\text{from } student\_grades B \\
\text{where } B.GPA > A.GPA)) \text{ as s\_rank} \\
\text{from } student\_grades A \\
\text{order by s\_rank;}
\]
GRANT and REVOKE keywords
  - grant select on instructor to $U_1, U_2, U_3$
  - revoke select on branch from $U_1, U_2, U_3$

Can provide select, insert, update, delete privileges

Can also create “Roles” and do security at the level of roles

Some databases support doing this at the level of individual “tuples”
  - PostgreSQL: https://www.postgresql.org/docs/10/ddl-rowsecurity.html
Fun with SQL

- https://blog.jooq.org/2016/04/25/10-sql-tricks-that-you-didnt-think-were-possible/
  - Long slide-deck linked off of this page
  - Complex SQL queries showing how to do things like: do Mandelbrot, solve subset sum problem etc.

- The MADlib Analytics Library or MAD Skills, the SQL; https://arxiv.org/abs/1208.4165

1. Everything is a Table

```sql
SELECT *
FROM (SELECT *
      FROM person
     ) t
```

Everything is a table. In PostgreSQL, even functions are tables:

```sql
SELECT *
FROM substring('abcde', 2, 3)
```
2. Recursion can be very powerful

WITH RECURSIVE t(v) AS ( 
    SELECT 1     -- Seed Row 
    UNION ALL 
    SELECT v + 1 -- Recursion 
    FROM t 
) 
SELECT v 
FROM t 
LIMIT 5

It yields

1
2
3
4
5

Makes SQL Turing-Complete

https://blog.jooq.org/2016/04/25/10-sql-tricks-that-you-didnt-think-were-possible/
3. Window Functions

```
SELECT depname, empno, salary, avg(salary) OVER (PARTITION BY depname) FROM empsalary;
```

<table>
<thead>
<tr>
<th>depname</th>
<th>empno</th>
<th>salary</th>
<th>avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>develop</td>
<td>11</td>
<td>5200</td>
<td>5020.0000000000000000000</td>
</tr>
<tr>
<td>develop</td>
<td>7</td>
<td>4200</td>
<td>5020.0000000000000000000</td>
</tr>
<tr>
<td>develop</td>
<td>9</td>
<td>4500</td>
<td>5020.0000000000000000000</td>
</tr>
<tr>
<td>develop</td>
<td>8</td>
<td>6000</td>
<td>5020.0000000000000000000</td>
</tr>
<tr>
<td>develop</td>
<td>10</td>
<td>5200</td>
<td>5020.0000000000000000000</td>
</tr>
<tr>
<td>personnel</td>
<td>5</td>
<td>3500</td>
<td>3700.0000000000000000000</td>
</tr>
<tr>
<td>personnel</td>
<td>2</td>
<td>3900</td>
<td>3700.0000000000000000000</td>
</tr>
<tr>
<td>sales</td>
<td>3</td>
<td>4800</td>
<td>4866.666666666666666667</td>
</tr>
<tr>
<td>sales</td>
<td>1</td>
<td>5000</td>
<td>4866.666666666666666667</td>
</tr>
<tr>
<td>sales</td>
<td>4</td>
<td>4800</td>
<td>4866.666666666666666667</td>
</tr>
</tbody>
</table>

(10 rows)
4. Correlation Coefficient

SET ARITHABORT ON;

DECLARE @OurData TABLE
(
  x NUMERIC(18,6) NOT NULL,
  y NUMERIC(18,6) NOT NULL
);

INSERT INTO @OurData
  (x, y)
SELECT
  x, y
FROM (VALUES
(1,32), (1,23), (3,50), (11,37), (-2,39), (10,44), (27,32), (25,16), (20,23),
(4,5), (30,41), (29,2), (31,52), (29,12), (50,40), (43,18), (10,65), (44,26),
(35,15), (24,37), (52,66), (59,46), (64,95), (79,36), (24,66), (69,58), (88,56),
(61,21), (100,60), (62,54), (10,14), (22,40), (52,97), (81,26), (37,58), (93,71),
(64,82), (24,33), (112,49), (64,90), (53,90), (132,61), (104,35), (60,52),
(29,50), (85,116), (95,104), (131,37), (139,38), (8,124)
) f(x,y)

SELECT
  ((Sy * Sxx) - (Sx * Sxy)) / ((N * (Sxx)) - (Sx * Sx)) AS a,
  ((N * Sxy) - (Sx * Sy)) / ((N * Sxx) - (Sx * Sx)) AS b,
  ((N * Sxy) - (Sx * Sy)) / SQRT
  (((N * Sxx) - (Sx * Sx)) * ((N * Syy - (Sy * Sy)))) AS r
FROM
SELECT SUM([@OurData].x) AS Sx, SUM([@OurData].y) AS Sy,
  SUM([@OurData].x * [@OurData].x) AS Sxx,
  SUM([@OurData].x * [@OurData].y) AS Sxy,
  SUM([@OurData].y * [@OurData].y) AS Syy,
  COUNT(*) AS N
FROM @OurData
) sums;
5. Page Rank

- Recursive algorithm to assign weights to the nodes of a graph (Web Link Graph)
- Weight for a node depends on the weights of the nodes that point to it
- Typically done in iterations till “convergence”
- Not obvious that you can do it in SQL, but:
  - Each iteration is just a LEFT OUTERJOIN
  - Stopping condition is trickier
- Other ways to do it as well

https://devnambi.com/2013/pagerank.html
declare @DampingFactor decimal(3,2) = 0.85 -- set the damping factor
    ,@MarginOfError decimal(10,5) = 0.001 -- set the stable weight
    ,@TotalNodeCount int
    ,@IterationCount int = 1

-- we need to know the total number of nodes in the system
    set @TotalNodeCount = (select count(*) from Nodes)

-- iterate!
    WHILE EXISTS
    ( -- stop as soon as all nodes have converged
        SELECT *
        FROM dbo.Nodes
        WHERE HasConverged = 0
    )
    BEGIN

        UPDATE n SET
        NodeWeight = 1.0 - @DampingFactor + isnull(x.TransferWeight, 0.0)

        -- a node has converged when its existing weight is the same as the weight it would be given
        -- (plus or minus the stable weight margin of error)
        ,HasConverged = case when abs(n.NodeWeight - (1.0 - @DampingFactor + isnull(x.TransferWeight, 0.0))) < @MarginOfError then 1
        else 0 end
        FROM Nodes n
        LEFT OUTER JOIN
        ( -- Here's the weight calculation in place
            SELECT e.TargetNodeId
            ,TransferWeight = sum(n.NodeWeight / n.NodeCount) * @DampingFactor
            FROM Nodes n
            INNER JOIN Edges e
            ON n.NodeId = e.SourceNodeId
            GROUP BY e.TargetNodeId
        ) as x
        ON x.TargetNodeId = n.NodeId

        -- for demonstration purposes, return the value of the nodes after each iteration
        SELECT @IterationCount as IterationCount
        ,*
        FROM Nodes

        set @IterationCount += 1
    END
Today’s Plan

- SQL (Chapter 3, 4) – Remaining Stuff

- Entity-Relationship Modeling
  - Entity-relationship Model (E/R model)
  - Converting from E/R to Relational

- Wednesday: Anatomy of a Web Application
  - Project 2
Entity-Relationship Model

Two key concepts

- **Entities:**
  - An object that *exists* and is *distinguishable* from other objects
  - Examples: Bob Smith, BofA, CMSC424
  - Have *attributes* (people have names and addresses)
  - Form *entity sets* with other entities of the same type that share the same properties
    - Set of all people, set of all classes
  - Entity sets may overlap
    - Customers and Employees
Entity-Relationship Model

- Two key concepts
  - **Relationships:**
    - Relate 2 or more entities
      - E.g. Bob Smith *has account at* College Park Branch
    - Form *relationship sets* with other relationships of the same type that share the same properties
      - Customers *have accounts at* Branches
    - Can have attributes:
      - *has account at* may have an attribute *start-date*
    - Can involve more than 2 entities
      - Employee *works at* Branch *at* Job
Entities and relationships

Two Entity Sets

<table>
<thead>
<tr>
<th>instructor</th>
<th>student</th>
</tr>
</thead>
<tbody>
<tr>
<td>76766</td>
<td>98988</td>
</tr>
<tr>
<td>45565</td>
<td>12345</td>
</tr>
<tr>
<td>10101</td>
<td>00128</td>
</tr>
<tr>
<td>98345</td>
<td>76543</td>
</tr>
<tr>
<td>76543</td>
<td>76653</td>
</tr>
<tr>
<td>22222</td>
<td>23121</td>
</tr>
<tr>
<td></td>
<td>44553</td>
</tr>
<tr>
<td>Crick</td>
<td>Tanaka</td>
</tr>
<tr>
<td>Katz</td>
<td>Shankar</td>
</tr>
<tr>
<td>Srinivasan</td>
<td>Zhang</td>
</tr>
<tr>
<td>Kim</td>
<td>Brown</td>
</tr>
<tr>
<td>Singh</td>
<td>Aoi</td>
</tr>
<tr>
<td>Einstein</td>
<td>Chavez</td>
</tr>
<tr>
<td></td>
<td>Peltier</td>
</tr>
</tbody>
</table>

Advisor Relationship, with and without attributes

<table>
<thead>
<tr>
<th>instructor</th>
<th>student</th>
</tr>
</thead>
<tbody>
<tr>
<td>76766</td>
<td>98988</td>
</tr>
<tr>
<td>45565</td>
<td>12345</td>
</tr>
<tr>
<td>10101</td>
<td>00128</td>
</tr>
<tr>
<td>98345</td>
<td>76543</td>
</tr>
<tr>
<td>76543</td>
<td>76653</td>
</tr>
<tr>
<td>22222</td>
<td>23121</td>
</tr>
<tr>
<td></td>
<td>44553</td>
</tr>
<tr>
<td>Crick</td>
<td>Tanaka</td>
</tr>
<tr>
<td>Katz</td>
<td>Shankar</td>
</tr>
<tr>
<td>Srinivasan</td>
<td>Zhang</td>
</tr>
<tr>
<td>Kim</td>
<td>Brown</td>
</tr>
<tr>
<td>Singh</td>
<td>Aoi</td>
</tr>
<tr>
<td>Einstein</td>
<td>Chavez</td>
</tr>
<tr>
<td></td>
<td>Peltier</td>
</tr>
</tbody>
</table>

3 May 2008
10 June 2007
12 June 2006
6 June 2009
30 June 2007
31 May 2007
4 May 2006
Alternative representation, used in the book in the past

Both notations used commonly
Rest of the class

- Details of the ER Model
  - How to represent various types of constraints/semantic information etc.

- Design issues

- A detailed example
Next: Relationship Cardinalities

- We may know:
  - One customer can only open one account
  - OR
  - One customer can open multiple accounts

- Representing this is important

- Why?
  - Better manipulation of data
    - If former, can store the account info in the customer table
  - Can enforce such a constraint
    - Application logic will have to do it; NOT GOOD
  - Remember: If not represented in conceptual model, the domain knowledge may be lost
Mapping Cardinalities

- Express the number of entities to which another entity can be associated via a relationship set
- Most useful in describing binary relationship sets
Mapping Cardinalities

- One-to-One
- One-to-Many
- Many-to-One
- Many-to-Many
Mapping Cardinalities

- Express the number of entities to which another entity can be associated via a relationship set

- Most useful in describing binary relationship sets

- N-ary relationships?
  - More complicated
  - Details in the book
Next: Types of Attributes

- Simple vs Composite
  - Single value per attribute?

- Single-valued vs Multi-valued
  - E.g. Phone numbers are multi-valued

- Derived
  - If date-of-birth is present, age can be derived
  - Can help in avoiding redundancy, enforcing constraints etc...
Types of Attributes

Primary key underlined: ID

Composite:
- name
- first_name
- middle_initial
- last_name
- address
  - street
  - street_number
  - street_name
  - apt_number
- city
- state
- zip
- { phone_number }
- date_of_birth
- age ()

Multi-valued:

Derived:
What attributes are needed to represent a relationship completely and uniquely?

- Union of primary keys of the entities involved, and relationship attributes

\{instructor.ID, date, student.ID\} describes a relationship completely
Is \{student\_id, date, instructor\_id\} a candidate key?

- No. Attribute \textit{date} can be removed from this set without losing key-ness
- In fact, union of primary keys of associated entities is always a superkey
Is \{\text{student\_id}, \text{instructor\_id}\} a candidate key?

- Depends

**Figure 7.8** E-R diagram with an attribute attached to a relationship set.
Is \{student\_id, instructor\_id\} a candidate key?

- Depends

If one-to-one relationship, either \{instructor\_id\} or \{student\_id\} sufficient

- Since a given instructor can only have one advisee, an instructor entity can only participate in one relationship
- Ditto student
Is \{\text{student}\_id, \text{instructor}\_id\} a candidate key?

- Depends

If one-to-many relationship (as shown), \{\text{student}\_id\} is a candidate key

- A given instructor can have many advisees, but at most one advisor per student allowed
General rule for binary relationships
  - one-to-one: primary key of either entity set
  - one-to-many: primary key of the entity set on the many side
  - many-to-many: union of primary keys of the associate entity sets

n-ary relationships
  - More complicated rules
What have we been doing

Why?

Understanding this is important
  ◦ Rest are details !!
  ◦ That’s what books/manuals are for.
Sometimes a relationship associates an entity set to itself

- Need “roles” to distinguish