CMSC 424 – Database design
Lecture 7
SQL, constraints

Mihai Pop

Admin issue

- Office hours tomorrow 10-11am
- Issues/concerns?

Homework 1...answers

- If grade < 7-8/10 you should worry!
- E-R diagram – on board
  - -1pt if links to “course” rather than “offering”
- Participatory constraints
  - course-offering in TOTAL participation with courses
  - -2pt if “total participation” or “partial participation” not mentioned in answer
- Constraints on course taking
  - -0.5 if constraints on wrong edge
  - -2pt if using attributes instead of constraints in E-R diagram
- File system
  - -1 pt if no E-R diagram
  - -1 pt if pseudocode does not specifically address how files laid out on disk and three operations not described
  - -1 pt if not specifically addressing multiple users accessing the datastructures

Aggregate Functions – Group By

- Find the number of depositors for each branch.

```
select branch_name, count (distinct customer_name)
        from depositor, account
        where depositor.account_number = account.account_number
        group by branch_name
```

Note: Attributes in **select** clause outside of aggregate functions must appear in **group by** list

Aggregate Functions – Having Clause

- Find the names of all branches where the average account balance is more than $1,200.

```
select branch_name, avg (balance)
       from account
       group by branch_name
       having avg (balance) > 1200
```

Note: predicates in the **having** clause are applied after the formation of groups whereas predicates in the **where** clause are applied before forming groups

Complex Queries using With Clause

- Find all branches where the total account deposit is greater than the average of the total account deposits at all branches.

```
with branch_total (branch_name, value) as
        select branch_name, sum (balance)
        from account
        group by branch_name
with branch_total_avg (value) as
        select avg (value)
        from branch_total
select branch_name
from branch_total, branch_total_avg
where branch_total.value >= branch_total_avg.value
```

Example Query

- Find all customers who have both an account and a loan at the bank.

```
select distinct customer_name
        from borrower
        where customer_name in (select customer_name
                                       from depositor )
```

- Find all customers who have a loan at the bank but do not have an account at the bank

```
select distinct customer_name
        from borrower
        where customer_name not in (select customer_name
                                            from depositor )
```

Set Comparison

- Find all branches that have greater assets than some branch located in Brooklyn.

```
select distinct  T.branch_name
        from branch as T, branch as S
        where  T.assets > S.assets and
               S.branch_city = 'Brooklyn'
```

- Same query using > **some** clause

```
select branch_name
        from branch
        where assets > some
                (select assets
                 from branch
                 where branch_city = 'Brooklyn')
```

Example Query

- Find the names of all branches that have greater assets than all branches located in Brooklyn.

```
select branch_name
        from branch
        where assets > all
                (select assets
                from branch
                where branch_city = 'Brooklyn')
```

Example Query

- Find all customers who have an account at all branches located in Brooklyn.

```
select distinct S.customer_name
        from depositor as S
        where not exists (
                (select branch_name
                from branch
                where branch_city = 'Brooklyn')
                except
                (select R.branch_name
                from depositor as T, account as R
                where T.account_number = R.account_number and
                        S.customer_name = T.customer_name ))
```

- Note that $X - Y = \varnothing \iff X \subseteq Y$
- *Note:* Cannot write this query using = **all** and its variants

temp tables, other...

- Select into

  ```
  select * into temp_table
  from ...
  ```

- Note that in SQL results are not sets – relational algebra must be redefined as BAG operations instead of SET operations

SQL: Summary

Clause	Eval Order	Semantics (RA/RA*)
`SELECT [(DISTINCT)]`	4	π (or π^*)
`FROM`	1	$\times^*$
`WHERE`	2	σ^*
`INTO`	7	$\leftarrow$
`GROUP BY`	3	Extended relational operator g
`HAVING`	5	σ^*
`ORDER BY`	6	Can’t express: requires ordered sets, bags
`AS`	-	ρ
`UNION ALL`	8	U*
`UNION`		U
`(similarly intersection, except)`		

Example Queries

- A view consisting of branches and their customers

```
create view all-customers as
  (select branch-name, customer-name
   from depositor, account
   where depositor.account-number = account.account-number)
    union
 (select branch-name, customer-name
  from borrower, loan
  where borrower.loan-number = loan.loan-number)
```

Find all customers of the Perryridge branch

```
select customer-name
             from all-customers
             where branch-name =  ‘Perryridge’
```

Views

- Is it different from DBMS's side ?
  - Yes; a view may or may not be *materialized*
  - Pros/Cons ?

- Updates into views have to be treated differently
  - In most cases, disallowed.

Modification of the Database – Updates

Increase all accounts with balances over $10,000 by 6%, all other accounts receive 5%.

Write two update statements:

```
update account
set balance = balance ∗ 1.06
where balance > 10000

update account
set balance = balance ∗ 1.05
where balance ≤ 10000
```

The order is important

Can be done better using the case statement

Modification of the Database – Deletion

Delete all account records at the Perryridge branch

```
delete from account
where branch-name = ‘Perryridge’
```

Delete all accounts at every branch located in Needham city.

```
delete from account
where branch-name in (select branch-name
                  from branch
                  where branch-city = ‘Needham’)
delete from depositor
where account-number in
            (select account-number
      from branch, account
      where branch-city = ‘Needham’
       and branch.branch-name = account.branch-name)
```

Example Query

Delete the record of all accounts with balances below the average at the bank.

```
delete from account
    where balance < (select avg (balance)
          from account)
```

Problem: as we delete tuples from *deposit,* the average balance changes

Solution used in SQL:

- First, compute **avg** balance and find all tuples to delete
- Next, delete all tuples found above (without recomputing **avg** or retesting the tuples)

Modification of the Database – Insertion

Add a new tuple to account

```
        insert into account
                values (‘A-9732’, ‘Perryridge’,1200)
```

or equivalently

```
    insert into account (branch-name, balance, account-number)
        values (‘Perryridge’, 1200, ‘A-9732’)
```

Add a new tuple to account with balance set to null

```
        insert into account
                values (‘A-777’,‘Perryridge’, null)
```

Update of a View

Create a view of all loan data in loan relation, hiding the amount attribute

```
create view branch-loan as
        select branch-name, loan-number
        from loan
```

Add a new tuple to branch-loan

```
insert into branch-loan
        values (‘Perryridge’, ‘L-307’)
```

This insertion must be represented by the insertion of the tuple

```
(‘L-307’, ‘Perryridge’, null)
```

into the loan relation

Updates on more complex views are difficult or impossible to translate, and hence are disallowed.

Many SQL implementations allow updates only on simple views (without aggregates) defined on a single relation

Modification of the Database – Updates

- Increase all accounts with balances over $10,000 by 6%, all other accounts receive 5%.
  - Write two **update** statements:

```
update account
set balance = balance ∗ 1.06
where balance > 10000


update account
set balance = balance ∗ 1.05
where balance ≤ 10000
```

  - The order is important
  - Can be done better using the **case** statement (next slide)

Case Statement for Conditional Updates

- Same query as before: Increase all accounts with balances over $10,000 by 6%, all other accounts receive 5%.

```
update account
set balance =  case
                  when balance <= 10000
                    then balance *1.05
                    else   balance * 1.06
                end
```

Next

NULLS

More SQL: Nulls

The “dirty little secret” of SQL

(major headache for query optimization)

Can be a value of any attribute

e.g: branch =

bname	**bcity**	**assets**
Downtown	Boston	9M
Perry	Horseneck	1.7M
Mianus	Horseneck	.4M
Waltham	Boston	NULL

What does this mean?

(unknown) We don’t know Waltham’s assets?

(inapplicable) Waltham has a special kind of account without assets

(withheld) We are not allowed to know

More SQL: Nulls

Arithmetic Operations with `Null`

`n + NULL = NULL` (similarly for all *arithmetic ops*: `+, -, *, /, mod,` …)

e.g: branch =

bname	bcity	assets
Downtown	Boston	9M
Perry	Horseneck	1.7M
Mianus	Horseneck	.4M
Waltham	Boston	NULL

```
SELECT bname, assets * 2 as a2
FROM branch
```

=

bname	a2
Downtown	18M
Perry	3.4M
Mianus	.8M
Waltham	NULL

More SQL: Nulls

Boolean Operations with `Null`

`n < NULL = UNKNOWN` (similarly for all *boolean ops*: `>, <=, >=, <>, =,` …)

e.g: branch =

bname	bcity	assets
Downtown	Boston	9M
Perry	Horseneck	1.7M
Mianus	Horseneck	.4M
Waltham	Boston	NULL

```
SELECT *
FROM branch
WHERE assets = NULL
```

=

bname	bcity	assets

Counter-intuitive: NULL * 0 = NULL

Counter-intuitive: select * from movies
where length >= 120 or length <= 120

More SQL: Nulls

Boolean Operations with `Null`

`n < NULL = UNKNOWN` (similarly for all *boolean ops*: `>, <=, >=, <>, =, ...`)

e.g: branch =

bname	bcity	assets
Downtown	Boston	9M
Perry	Horseneck	1.7M
Mianus	Horseneck	.4M
Waltham	Boston	NULL

```
SELECT *
FROM branch
WHERE assets IS NULL
```

=

bname	bcity	assets
Waltham	Boston	NULL

Transactions

A transaction is a sequence of queries and update statements executed as a single unit

Transactions are started implicitly and terminated by one of

- commit work: makes all updates of the transaction permanent in the database
- rollback work: undoes all updates performed by the transaction.

Motivating example

Transfer of money from one account to another involves two steps:

- deduct from one account and credit to another

If one steps succeeds and the other fails, database is in an inconsistent state

Therefore, either both steps should succeed or neither should

If any step of a transaction fails, all work done by the transaction can be undone by rollback work.

Rollback of incomplete transactions is done automatically, in case of system failures

Transactions (Cont.)

In most database systems, each SQL statement that executes successfully is automatically committed.

- *Each transaction would then consist of only a single statement*
- *Automatic commit can usually be turned off, allowing multi-statement transactions, but how to do so depends on the database system*
- *Another option in SQL:1999: enclose statements within*

```
begin atomic
   …
end
```

Triggers

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

Trigger Example

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*

Trigger Example in SQL:1999

```
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
```

Trigger Example in SQL:1999

```
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…