CS 540
Database Management Systems

Review of Relational Model and SQL

Oregon State University

Review problems:
people betting on OSU football games

Out(<u>game</u>, <u>outcome</u>) Bets(<u>who</u>, <u>outcome</u>, <u>game</u>, amt)

game	outcome
USC	W
UCLA	L
Stanford	W

who	outcome	game	amt
John	W	USC	200
John	W	UCLA	100
John	L	Arizona	150
Kevin	L	UO	210
Kevin	L	UCLA	50
Kevin	W	Stanford	120

Some games have not been played yet, e.g., *Arizona*.

Problem 1

List the completed games that nobody bet on.

game	outcome
USC	W
UCLA	L
Stanford	W

who	outcome	game	amt
John	W	USC	200
John	W	UCLA	100
John	L	Arizona	150
Kevin	L	UO	210
Kevin	L	UCLA	50
Kevin	W	Stanford	120

game

Problem 1

List the completed games that nobody bet on.

```
(Select Game
 From Out)
Except
 (Select Game
  From Bets)
```

Problem 2

Who bet the most money on a single game?

game	outcome
USC	W
UCLA	L
Stanford	W

who	outcome	game	amt
John	W	USC	200
John	W	UCLA	100
John	L	Arizona	150
Kevin	L	UO	210
Kevin	L	UCLA	50
Kevin	W	Stanford	120

who	amt
Kevin	210

Oregon State University

Problem 2

Who bet the most money on a single game?

```
Select Who, Amt
From Bets
Where Amt >= All
                (Select Amt
                 From Bets)
```

Problem 3

List the games that all bettors agree on.

game	outcome
USC	W
UCLA	L
Stanford	W

who	outcome	game	amt
John	W	USC	200
John	W	UCLA	100
John	L	Arizona	150
Kevin	L	UO	210
Kevin	L	UCLA	50
Kevin	W	Stanford	120

game
Stanford
Arizona
UO
USC

OSU
Oregon State
UNIVERSITY

Problem 3

List the games that all bettors agree on.

```
(Select game
 From Bets)
 Except
(Select Bets1.game
 From Bets Bets1, Bets Bets2
 Where (Bets1.game = Bets2.game)
   And (Bets1.outcome <> Bets2.outcome))
```

Problem 4

For each game, the number of people betting on OSU to win and the number betting on OSU to lose.

game	outcome
USC	W
UCLA	L
Stanford	W

game	outcome	num
Stanford	W	1
UO	L	1
UCLA	W	1
UCLA	L	1
USC	W	1
Arizona	L	1

who	outcome	game	amt
John	W	USC	200
John	W	UCLA	100
John	L	Arizona	150
Kevin	L	UO	210
Kevin	L	UCLA	50
Kevin	W	Stanford	120

OSU
Oregon State
UNIVERSITY

Problem 4

For each game, the number of people betting on OSU to win and the number betting on OSU to lose.

```
Select game, outcome, Count(who) As num
From Bets
Group By game, outcome
```

Problem 5

Find the people who have made two or more bets on OSU to lose.

game	**outcome**
USC	W
UCLA	L
Stanford	W

who	**outcome**	**game**	**amt**
John	W	USC	200
John	W	UCLA	100
John	L	Arizona	150
Kevin	L	UO	210
Kevin	L	UCLA	50
Kevin	W	Stanford	120

who
Kevin

Oregon State University

Problem 5

Find the people who have made two or more bets on OSU to lose.

```
Select who
From Bets
Where outcome = 'L'
Group By who
Having Count(outcome) >= 2
```

Problem 6

Who bet the most money overall?

game	outcome
USC	W
UCLA	L
Stanford	W

who	sumAmt
John	450

who	outcome	game	amt
John	W	USC	200
John	W	UCLA	100
John	L	Arizona	150
Kevin	L	UO	210
Kevin	L	UCLA	50
Kevin	W	Stanford	120

OSU Oregon State UNIVERSITY

Problem 6

Who bet the most money overall?

```
Select who, Sum(amt) As sumAmt
From Bets
Group By who
Having Sum(amt) >= ALL
                    (Select Sum(amt)
                      From Bets
                      Group By who)
```

Problem 7

Who has bet on every game?

game	outcome
USC	W
UCLA	L
Stanford	W

who	outcome	game	amt
John	W	USC	200
John	W	UCLA	100
John	L	Arizona	150
Kevin	L	UO	210
Kevin	L	UCLA	50
Kevin	W	Stanford	120

Who

OSU Oregon State UNIVERSITY

Problem 7

Who has bet on every game?

```
Create View AllGames As
(Select  game From Out) Union
(Select game From Bets)

Select who
From Bets
Group By who
Having Count(Distinct game) =
(Select Count(Distinct game)
 From AllGames)
```

Problem 8

What games have won the most money for the people who bet on OSU to win?

game	outcome
USC	W
UCLA	L
Stanford	W

game
USC

who	outcome	game	amt
John	W	USC	200
John	W	UCLA	100
John	L	Arizona	150
Kevin	L	UO	210
Kevin	L	UCLA	50
Kevin	W	Stanford	120

Oregon State University

Problem 8

What games have won the most money for the people who bet on OSU to win?

```
Create View Success-Win As
 (Select Bets.game, Sum(Bets.amt) As SumAmt
  From Bets, Out
  Where out.game = Bets.game And
        Bets.outcome = 'W' And Out.outcome = 'W'
         Group By Game)

Select Distinct game
From Success-Win
Where SumAmt >= All
                 (Select SumAmt
                  From Success-Win)
```

Problem 9

List the people who won some money so far.

game	outcome
USC	W
UCLA	L
Stanford	W

who
John
Kevin

who	outcome	game	amt
John	W	USC	200
John	W	UCLA	100
John	L	Arizona	150
Kevin	L	UO	210
Kevin	L	UCLA	50
Kevin	W	Stanford	120

OSU Oregon State UNIVERSITY

Problem 9

List the people who won some money so far.

```
Create View Success As
(Select who, Sum(amt) As pAmt
 From Bets, Out
 Where Out.game = Bets.game And Bets.outcome = Out.outcome
 Group By who)

Create View Failure As
(Select who, Sum(amt) As nAmt
 From Bets, Out
 Where Out.game = Bets.game And Bets.outcome <> Out.outcome
 Group By who)
```

Problem 9

List the people who won some money so far.

```
(Select Success.who
 From Success, Failure
 Where Success.who = Failure.who And
       Success.pAmt > Failure.nAmt )

Union
 (Select who
   From Success
    Where who not in
                        (Select who
                         From Failure) )
```

Views

- Virtual relations defined over *base relations*

```
Create View <name> As <query>;
```

- *PopularShop(sname, addr):* shops with at least 10 frequent drinkers.

```
Create View PopularShop As
   Select C.sname, C.addr
   From CoffeeShop C, Frequents F
   Where C.sname = F.sname
    Group By C.sname, C.addr
    Having Count(dname) > 9
```

- Used in queries like the base relations.

```
Select addr
  From PopularShops;
```

Advantages of using views

- **Usability**
  - easier for users to find relevant relations.
  - DBAs create the relevant views for each group of users.
- **Security**
  - each group of users should access only a subset of database.
  - DBAs define views and their access privileges.
- **Efficiency**
  - One may precompute and store frequently used views.

Virtual versus materialized views

- **Virtual view**
  - database system stores the view definitions
  - database system rewrites the query using view definitions
- **Materialized view**
  - Database system pre-computes the views
    - Users issue many queries using *PopularShops*
  - **advantage:** improves the running time for some queries.
  - **disadvantage:** database system should refresh it if its base relations are modified.

Oregon State UNIVERSITY

Which views to materialize?

- **Tradeoff:** #read from view vs. #write to base relations
- Two types of database systems
  – Online Transaction Processing (OLTP)
    - banking systems, online shopping, …
    - heavy write workload
  – Online Analytics Processing (OLAP)
    - integrated database about branches
    - data warehouses
    - write is not frequent, e.g., once a month
    - many complex join queries with aggregation functions
- Materialized views are popular in data warehouses
  – no significant overhead

Maintaining materialized views with both r/w

- No silver bullet!
  - choose a policy that matches the application needs.
- Two steps
  - **Propagate:** compute changes to the view when data changes.
  - **Refresh:** apply changes to the materialized view table.
- Maintenance policy
  - **Immediate:** as soon as the base tables are updated.
    - **advantage:** views are always consistent.
    - **disadvantage:** updates are slower.
  - **Deferred:** some time later
    - **advantage:** scale to many views without slowing down updates .
    - **disadvantage:** views may become inconsistent.

Oregon State UNIVERSITY

Deferred maintenance

- Lazy
  - Delay refresh until the next query over the view
  - Refresh before answering the query
- Periodic (snapshot)
  - refresh the view periodically
  - **disadvantage:** Some queries may be answered over an outdated version of view.
  - **advantage:** scale to update many views
  - more widely used

Support for view materialization

- Example in Oracle

```
Create materialized view PopularShop As
    Select sname, addr
    From CoffeeShop C, Frequents F
    Where C.sname = F.sname
     Group By C.sname, C.addr
     Having Count(dname) > 9
```

  - Also called *snapshot.*
- No direct support in MySQL
  - programmers have to implement it

View selection

- Given a query workload, find the minimum set of views to materialize such that
  - minimizes the running time of all queries
- General approach
  - find the time-consuming joins /aggregation sub-queries.
  - pick the ones that are used in sufficiently many times.
- Some database systems provide wizard programs to help

Is SQL Sufficient?

- Using IsParent(parent,child), find grand children of a given person.

- Now find all descendants of a given person.

- It is **not** possible to write this query in standard SQL!
  - We can prove it.
  - *3 bonus points for everybody who finds, understands and writes the main ideas of the proof and submits it by 2/1.*

Recursive queries in SQL

- SQL standards have recursive SQL
  - most database systems do not implement that
- Database systems usually support limited recursion
  - MySQL recursive cte, Oracle's connected by, …
- They define within a single query
  - a base case (base query)
  - a recursion step
- Systems limit the type of queries used for recursion
  - not group by/ aggregation function
  - to keep the plan for normal queries fast.

OSU Oregon State UNIVERSITY

Recursive queries in MySQL

- Common table expression (CTE)
  – relation variable within the scope of a single query

```
WITH cte (col1, col2) AS
(
    SELECT 1, 2
    UNION ALL
    SELECT 3, 4
)
SELECT col1, col2 FROM cte;
```

Oregon State University

Recursive queries in MySQL

- Common table expression (CTE)

```
WITH
  cte1 AS (SELECT a, b FROM table1),
  cte2 AS (SELECT c, d FROM table2)
SELECT b, d
  FROM cte1 JOIN cte2
  WHERE cte1.a = cte2.c;
```

- used similar to virtual view in the query
  - the query plan may be more efficient as each CTE is executed only once and used multiple times.

Recursive queries in MySQL

- Recursive CTE

```
WITH RECURSIVE cte (n) AS
(
    SELECT 1
    UNION ALL
    SELECT n + 1 FROM cte WHERE n < 5
)
SELECT * FROM cte;
```

base case (base query) ?

recursion step ?

Recursive queries in MySQL

- Using *Employee(id, name, manager_id)* produce the organizational chart of the management chain.

```
WITH RECURSIVE employee_paths (id, name, path) AS
(
   SELECT id, name, CAST(id AS CHAR(200))
   FROM employees WHERE manager_id IS NULL
   UNION ALL
   SELECT e.id, e.name, CONCAT(ep.path, ',', e.id)
   FROM employee_paths AS ep
   JOIN employees AS e ON ep.id = e.manager_id
)
SELECT * FROM employee_paths ORDER BY path;
```

Oregon State University