Introduction to SQL

Dr. Shrutilipi Bhattacharjee, Assistant Professor, Dept. of IT, NIT Karnataka, India

String Operations

- SQL includes a string-matching operator for comparisons on character strings

- The operator **like** uses patterns that are described using two special characters:
  - **Percent (%):** The % character matches any substring
  - **Underscore (_):** The _ character matches any character

- Find the names of all instructors whose name includes the substring “dar”

select *name*
from *instructor*
where *name* **like** **‘**%dar%**’**

- Match the string “100%” in that above we use backslash (\) as the escape character

like '100 \%' **escape** '\'

String Operations

- Patterns are case sensitive

- Pattern matching examples:
  - 'Intro%' matches any string beginning with “Intro”
  - '%Comp%' matches any string containing “Comp” as a substring
  - '_ _ _' matches any string of exactly three characters
  - '_ _ _ %' matches any string of at least three characters

- SQL supports a variety of string operations such as:
  - Concatenation (using “||”)
  - Converting from upper to lower case (and vice versa)
  - Finding string length, extracting substrings, etc.

Ordering the Display of Tuples

- List in alphabetic order the names of all instructors

  select distinct *name*
  from *instructor*
  order by *name*

- We may specify **desc** for descending order or **asc** for ascending order, for each attribute
- Ascending order is the default
  - Example: **order by** *name* **desc**

- Can sort on multiple attributes
  - Example: **order by** *dept_name, name*

Where Clause Predicates

- SQL includes a **between** comparison operator

- **Example:** Find the names of all instructors with salary between $90,000 and $100,000 (that is, ≥ $90,000 and ≤ $100,000)

select *name*
from *instructor*
where *salary* **between** 90000 **and** 100000

- Tuple comparison

select *name*, *course_id*
from *instructor*, *teaches*
where (*instructor.ID*, *dept_name*) = (*teaches.ID*, 'Biology');

Duplicates

- In relations with duplicates, SQL can define how many copies of tuples appear in the result

- *Multiset* versions of some of the relational algebra operators – given multiset relations r_1 and r_2:
  - $\sigma_\theta(r_1)$**:** If there are c_1 copies of tuple t_1 in r_1, and t_1 satisfies selections σ_θ, then there are c_1 copies of t_1 in $\sigma_\theta(r_1)$
  - $\Pi_A(r)$**:** For each copy of tuple t_1 in r_1, there is a copy of tuple $\Pi_A(t_1)$ in $\Pi_A(r_1)$ where $\Pi_A(t_1)$ denotes the projection of the single tuple t_1
  - $r_1 \times r_2$ **:** If there are c_1 copies of tuple t_1 in r_1 and c_2 copies of tuple t_2 in r_2, there are $c_1 \times c_2$ copies of the tuple $t_1 . t_2$ in $r_1 \times r_2$

Duplicates

- Example: Suppose multiset relations r_1 (*A, B*) and r_2 (*C*) are as follows:

  r_1 = {(1, *a*) (2, *a*)} r_2 = {(2), (3), (3)}

- Then $\Pi_B(r_1)$ would be {(a), (a)}, while $\Pi_B(r_1)$ x r_2 would be

  {(*a*, 2), (*a*, 2), (*a*, 3), (*a*, 3), (*a*, 3), (*a*, 3)}

- SQL duplicate semantics:

  select $A_1, A_2, ..., A_n$
  from $r_1, r_2, ..., r_m$
  where *P*

is equivalent to the *multiset* version of the expression: $\Pi_{A1,, A2, ..., An}(\sigma_P (r_1 \text{ x } r_2 \text{ x } ... \text{ x } r_m))$

Set Operations

- Find courses that ran in Fall 2017 or in Spring 2018

  (**select** *course_id* **from** *section* **where** *sem* = 'Fall' **and** *year* = 2017)
  union
  (**select** *course_id* **from** *section* **where** *sem* = 'Spring' **and** *year* = 2018)

- Find courses that ran in Fall 2017 and in Spring 2018

  (**select** *course_id* **from** *section* **where** *sem* = 'Fall' **and** *year* = 2017)
  intersect
  (**select** *course_id* **from** *section* **where** *sem* = 'Spring' **and** *year* = 2018)

- Find courses that ran in Fall 2017 but not in Spring 2018

  (**select** *course_id* **from** *section* **where** *sem* = 'Fall' **and** *year* = 2017)
  except
  (**select** *course_id* **from** *section* **where** *sem* = 'Spring' **and** *year* = 2018)

Set Operations

- Find the salaries of all the ***instructor***s that are less than the largest salary

  select distinct *T.salary*
  from *instructor* ***as*** *T, instructor* ***as*** *S*
  where *T.salary < S.salary*

- Find the salaries of all the ***instructor***s

  select distinct *salary*
  from *instructor*

- Find the largest salaries of all ***instructor***s

  select *(“second query”)*
  except
  select *(“first query”)*

Set Operations

- Set operations **union, intersect,** and **except**
  - Each of the above operations automatically eliminates duplicates

- To retain all duplicates use the corresponding multiset versions **union all, intersect all** and **except all**

- Suppose a tuple occurs *m* times in *r* and *n* times in *s,* then, it occurs:
  - *m* + *n* times in *r* **union all** *s*
  - min(*m,n)* times in *r* **intersect all** *s*
  - max(0, *m* – *n)* times in *r* **except all** *s*

Null Values

- It is possible for tuples to have a null value, denoted by **null**, for some of their attributes
- **null** signifies an unknown value or that a value does not exist
- The result of any arithmetic expression involving **null** is **null**
  - Example: 5 + **null** returns **null**
- The predicate **is null** can be used to check for null values
  - Example: Find all instructors whose salary is null

    select *name*
    from *instructor*
    where *salary* **is null**

- The predicate **is not null** succeeds if the value on which it is applied is not null

Null Values

- Three values – *true, false, unknown*
- Any comparison with *null* returns *unknown*
  - *E.g. 5 < null or null <> null or null = null*
- Three-valued logic using the truth value *unknown*:
  - OR: (*unknown* **or** *true*) = *true*, (*unknown* **or** *false*) = *unknown*
    (*unknown* **or** *unknown) = unknown*
  - AND: *(true* **and** *unknown) = unknown, (false* **and** *unknown) = false,*
    (unknown **and** *unknown) = unknown*
  - NOT*: (***not** *unknown) = unknown*
  - "*P* **is unknown"** evaluates to true if predicate *P* evaluates to *unknown*
- Result of **where** clause predicate is treated as *false* if it evaluates to *unknown*

Aggregate Functions

- These functions operate on the multiset of values of a column of a relation, and return a value

avg: average value
min: minimum value
max: maximum value
sum: sum of values
count: number of values

Aggregate Functions Examples

- Find the average salary of instructors in the Computer Science department

  select avg (*salary*)
  from *instructor*
  where *dept_name* = 'Comp. Sci.';

- Find the total number of instructors who teach a course in the Spring 2018 semester

  select count (**distinct** *ID*)
  from *teaches*
  where *semester* = 'Spring' **and** *year* = 2018;

- Find the number of tuples in the *course* relation

  select count (*)
  from *course*;

Aggregate Functions – *Group By*

- Find the average salary of instructors in each department

select *dept_name*, **avg** (*salary*) **as** *avg_salary*
from *instructor*
group by *dept_name*;

ID	*name*	*dept_name*	*salary*
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

dept_name	*avg_salary*
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

Aggregation

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

/* erroneous query */

select *dept_name*, *ID*, **avg** (*salary*)
from *instructor*
group by *dept_name*;

Aggregate Functions – *Having* Clause

- Find the names and average salaries of all departments whose average salary is greater than 42000

select *dept_name*, **avg** (*salary*) **as** *avg_salary*
from *instructor*
group by *dept_name*
having avg (*salary*) > 42000;

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

Null Values and Aggregates

- Total all salaries

select sum (*salary)*
from *instructor*

- Above statement ignores *null* amounts
- Result is *null* if there is no non-null amount
- All aggregate operations except **count(*)** ignore tuples with null values on the aggregated attributes
- What if collection has only *null* values?
  - count returns 0
  - All other aggregates return *null*

Next Lecture

Introduction to SQL

Thank you for your attention...

Any question?

Contact:
Department of Information Technology, NITK Surathkal, India
6th Floor, Room: 13
Phone: +91-9477678768
E-mail: shrutilipi@nitk.edu.in