Module - II

SQL - Structured Query Language

Lecture - 4

Data Query Language (DQL)

SQL - DQL

- Basic Query Structure
  - ✓ Syntax
  - ✓ The select Clause
  - ✓ The where Clause
  - ✓ The from Clause
  - ✓ Examples
  - ✓ Rename Operation
  - ✓ String Operations
  - ✓ Ordering the Display of Tuples
  - ✓ Where Clause Predicates

Data Query Language (DQL)

Data Query Language (DQL) is used to fetch the data from the database. It uses only one command:

1. **SELECT:** It is used to retrieve data from the database.

Basic Query Structure

- A typical SQL query has the form:

$$\textbf{select } A_1, A_2, \ldots, A_n$$
$$\textbf{from } r_1, r_2, \ldots, r_m$$
$$\textbf{where } P$$

  - A_i represents an attribute
  - R_i represents a relation
  - P is a predicate.

- The result of an SQL query is a relation.

The select Clause

- The **select** clause lists the attributes desired in the result of a query
  - corresponds to the projection operation of the relational algebra
- Example: find the names of all instructors:

```
select name
from instructor;
```

- NOTE: SQL names are case insensitive (i.e., you may use upper- or lower-case letters.)
  - E.g., *Name* ≡ *NAME* ≡ *name*
  - Some people use upper case wherever we use bold font.

The select Clause (Cont.)

- SQL allows duplicates in relations as well as in query results.
- To force the elimination of duplicates, insert the keyword **distinct** after select.
- Find the department names of all instructors, and remove duplicates

```
select distinct dept_name
from instructor;
```

dept_name
Comp. Sci.
Finance
Music
Physics
History
Physics
Comp. Sci.
History
Finance
Biology
Comp. Sci.
Elec. Eng.

- The keyword **all** specifies that duplicates should not be removed.

```
select all dept_name
from instructor;
```

The select Clause (Cont.)

- An asterisk in the select clause denotes "all attributes"

```
select * from instructor
```

- An attribute can be a literal with no **from** clause

```
select '437'
```

  - Results is a table with one column and a single row with value "437"
  - Can give the column a name using:

```
select '437' as ID
```

- An attribute can be a literal with **from** clause

```
select 'A' from instructor
```

  - Result is a table with one column and *N* rows (number of tuples in the *instructors* table), each row with value "A"

The select Clause (Cont.)

- The **select** clause can contain arithmetic expressions involving the operation, +, –, *, and /, and operating on constants or attributes of tuples.
  - ✓ The query:

```
select ID, name, salary/12 from instructor
```

would return a relation that is the same as the *instructor* relation, except that the value of the attribute *salary* is divided by 12.

- Can rename "*salary/12*" using the **as** clause:

```
select ID, name, salary/12 as monthly_salary
```

The where Clause

- The **where** clause specifies conditions that the result must satisfy
  - Corresponds to the selection predicate of the relational algebra.
- Example: *To find all instructors in Comp. Sci. dept*

```
select name
from instructor
where dept_name = 'Comp. Sci.'
```

The where Clause

- SQL allows the use of the logical connectives **and**, **or**, and **not**
- The operands of the logical connectives can be expressions involving the comparison operators <, <=, >, >=, =, and <>.
- Comparisons can be applied to results of arithmetic expressions

Example: *To find all instructors in Comp. Sci. dept with salary > 70000*

```
select name
from instructor
where dept_name = 'Comp. Sci.'  and salary > 70000
```

The from Clause

- The from clause lists the relations involved in the query
  - Corresponds to the Cartesian product operation of the relational algebra.

Example: *Find the Cartesian product instructor X teaches*

```
select * from instructor, teaches
```

  - Generates every possible instructor – teaches pair, with all attributes from both relations.
  - For common attributes (e.g., `ID`), the attributes in the resulting table are renamed using the relation name (e.g., `instructor.ID`)

- Cartesian product not very useful directly, but useful combined with where-clause condition (selection operation in relational algebra).

Examples

Example-1: *Find the names of all instructors who have taught some course and the course_id*

```
select name, course_id
from instructor , teaches
where instructor.ID = teaches.ID
```

name	*course_id*
Srinivasan	CS-101
Srinivasan	CS-315
Srinivasan	CS-347
Wu	FIN-201
Mozart	MU-199
Einstein	PHY-101
El Said	HIS-351
Katz	CS-101
Katz	CS-319
Crick	BIO-101
Crick	BIO-301
Brandt	CS-190
Brandt	CS-190
Brandt	CS-319
Kim	EE-181

Example-2: *Find the names of all instructors in the Art department who have taught some course and the course_id*

```
select name, course_id
from instructor , teaches
where instructor.ID = teaches.ID
      and  instructor. dept_name = 'Art'
```

The Rename Operation

- The SQL allows renaming *relations* and *attributes* using the as clause:

```
old-name as new-name
```

Example: *Find the names of all instructors who have a higher salary than some instructor in 'Comp. Sci'.*

```
select distinct T.name
from instructor as T, instructor as S
where T.salary > S.salary and S.dept_name =
'Comp. Sci.'
```

Keyword **as** is optional and may be omitted

```
instructor as T ≡ instructor T
```

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.

Example: *Find the names of all instructors whose name includes the substring "dar".*

```
select name
from instructor
where name like '%dar%'
```

String Operations

- Match the string “100%”

```
like '100 \%'  escape  '\'
```

in that above we use backslash (\) as the escape character.

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.

String Operations

- 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:

```
select distinct name
from    instructor
order by name desc
```

Ordering the Display of Tuples (cont'd)

- Can sort on multiple attributes
- Example:

```
select distinct name
from    instructor
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
```

Where Clause Predicates (cont'd)

- Tuple comparison
- Example:

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