CSCC43H: Introduction to Databases

Lecture 6

Wael Aboulsaadat

Acknowledgment: these slides are partially based on Prof. Garcia-Molina & Prof. Ullman slides accompanying the course's textbook.

Database Management System (DBMS)

- A collection of programs that enable:

  Defining (describing the structure),

  Populating by data (Constructing),

  → Manipulating (querying, updating),

  – Preserving consistency,
  – Protecting from misuse,
  – Recovering from failure, and
  – Concurrent using

  of a database.

Example Database

EMPLOYEE

FirstName	Surname	Dept	Office	Salary	City
Mary	Brown	Administration	10	45	London
Charles	White	Production	20	36	Toulouse
Gus	Green	Administration	20	40	Oxford
Jackson	Neri	Distribution	16	45	Dover
Charles	Brown	Planning	14	80	London
Laurence	Chen	Planning	7	73	Worthing
Pauline	Bradshaw	Administration	75	40	Brighton
Alice	Jackson	Production	20	46	Toulouse

DEPARTMENT

DeptName	Address	City
Administration	Bond Street	London
Production	Rue Victor Hugo	Toulouse
Distribution	Pond Road	Brighton
Planning	Bond Street	London
Research	Sunset Street	San José

Duplicates

- In the relational algebra and calculus the results of queries do not contain duplicates.
- In SQL, tables may have identical rows.
- Duplicates can be removed using the keyword ***distinct***:

```
select City
from Department
```

City
London
Toulouse
Brighton
London
San José

```
select distinct City
from Department
```

City
London
Toulouse
Brighton
San José

Joins

- SQL-2 introduced an alternative syntax for the representation of joins, representing them explicitly in the **`from`** clause:

`select` *AttrExpr* [[**`as`**] *Alias*] {, *AttrExpr* [[**`as`**] *Alias* **`from`**
Table [[**`as`**] *Alias*]
{[*JoinType*] **`join`** *Table*
[[**`as`**] *Alias*] **`on`** *JoinConditions* }
[**`where`** *OtherCondition*]

- *JoinType* can be any of **`inner`**, **`right [outer]`**, **`left [outer]`** or **`full [outer]`**.

Inner Join in SQL

- "Find the names of the employees and the cities in which they work":

```
select FirstName, Surname, D.City
from Employee inner join Department as D
     on Dept = DeptName
```

- Result:

FirstName	Surname	City
Mary	Brown	London
Charles	White	Toulouse
Gus	Green	London
Jackson	Neri	Brighton
Charles	Brown	London
Laurence	Chen	London
Pauline	Bradshaw	London
Alice	Jackson	Toulouse

Another Example: Drivers and Cars

DRIVER

FirstName	Surname	DriverID
Mary	Brown	VR 2030020Y
Charles	White	PZ 1012436B
Marco	Neri	AP 4544442R

AUTOMOBILE

CarRegNo	Make	Model	DriverID
ABC 123	BMW	323	VR 2030020Y
DEF 456	BMW	Z3	VR 2030020Y
GHI 789	Lancia	Delta	PZ 1012436B
BBB 421	BMW	316	MI 2020030U

Left Join

- "Find all drivers and their cars, if any":

```
select FirstName,Surname,
  Driver.DriverID,CarRegNo,Make,Model
from Driver left join Automobile on
  (Driver.DriverID = Automobile.DriverID)
```

- Result:

FirstName	Surname	DriverID	CarRegNo	Make	Model
Mary	Brown	VR 2030020Y	ABC 123	BMW	323
Mary	Brown	VR 2030020Y	DEF 456	BMW	Z3
Charles	White	PZ 1012436B	GHI 789	Lancia	Delta
Marco	Neri	AP 4544442R	NULL	NULL	NULL

Full Join

- "Find all possible drivers and their cars":

```
select FirstName,Surname,Driver.DriverID
       CarRegNo, Make, Model
from Driver full join Automobile on
  (Driver.DriverID = Automobile.DriverID)
```

- Result:

FirstName	Surname	DriverID	CarRegNo	Make	Model
Mary	Brown	VR 2030020Y	ABC 123	BMW	323
Mary	Brown	VR 2030020Y	DEF 456	BMW	Z3
Charles	White	PZ 1012436B	GHI 789	Lancia	Delta
Marco	Neri	AP 4544442R	NULL	NULL	NULL
NULL	NULL	NULL	BBB 421	BMW	316

The `order by` Clause

- ***`order by`*** — appearing at the end of a query — orders the rows of the result; syntax:

  `order by` *OrderingAttribute [* ***`asc`*** *|* ***`desc`*** *]*
  {, OrderingAttribute [***`asc`*** *|* ***`desc`*** *] }*

- Extract the content of the **`Automobile`** table in descending order with respect to make and model:

```
select * from Automobile
order by Make desc, Model desc
```

- Result:

CarRegNo	Make	Model	DriverID
GHI 789	Lancia	Delta	PZ 1012436B
DEF 456	BMW	Z3	VR 2030020Y
ABC 123	BMW	323	VR 2030020Y
BBB 421	BMW	316	MI 2020030U

Aggregate Queries

- The result of an aggregate query depends on functions that take as an argument a set of tuples.
- SQL-2 offers five aggregate operators:
  - **`count`**
  - **`sum`**
  - **`max`**
  - **`min`**
  - **`avg`**

Operator `count`

- **`count`** returns the number of elements (or, distinct elements) of its argument:

  `count`(< * | [**`distinct`** | **`all`**] *AttributeList* >)

- "Find the number of employees":

```
select count(*)from Employee
```

- "Find the number of different values on attribute Salary for all tuples in Employee":

```
select count(distinct Salary)
       from Employee
```

- "Find the number of tuples in Employee having non-null values on the attribute Salary":

```
select count(all Salary) from Employee
```

Sum, Average, Maximum and Minimum

- Syntax:
  < **sum** | **max** | **min** | **avg** > ([**distinct** | **all**]
  AttributeExpr)
- "Find the sum of all salaries for the Administration department":

```
select sum(Salary) as SumSalary
from Employee
where Dept = 'Administration'
```

- Result:

SumSalary
125

Aggregate Queries with Join

- "Find the maximum salary among the employees who work in a department based in London":

```
select max(Salary) as MaxLondonSal
from Employee, Department
where Dept = DeptName and
      Department.City = 'London'
```

- Result:

MaxLondonSal
80

Aggregate Queries and Target List

- Find the maximum and minimum salaries among all employees:

```
select max(Salary) as MaxSal,
       min(Salary) as MinSal
from Employee
```

- Result:

MaxSal	MinSal
80	36

Group by Queries

- Queries may apply aggregate operators to subsets of rows.
- "Find the sum of salaries of all the employees of the same department":

```
select Dept, sum(Salary) as TotSal
from Employee
group by Dept
```

- Result:

Dept	TotSal
Administration	125
Distribution	45
Planning	153
Production	82

Semantics of `group by` Queries - I

- First, the query is executed without **`group by`** and without aggregate operators:

```
select Dept, Salary
from Employee
```

Dept	Salary
Administration	45
Production	36
Administration	40
Distribution	45
Planning	80
Planning	73
Administration	40
Production	46

Semantics of `group by` Queries - II

- … then the query result is divided in subsets characterized by the same values for the attributes appearing as argument of the **`group by`** clause (in this case attribute Dept):
- Finally, the aggregate operator is applied separately to each subset

Dept	Salary
Administration	45
Administration	40
Administration	40
Distribution	45
Planning	80
Planning	73
Production	36
Production	46

Dept	TotSal
Administration	125
Distribution	45
Planning	153
Production	82

Group Predicates

- When conditions are defined on the result of an aggregate operator, it is necessary to use the **having** clause
- "Find which departments spend more than 100 on salaries":

```
select Dept
from Employee
group by Dept
having sum(Salary) > 100
```

- Result:

Dept
Administration
Planning

where or having?

- Only predicates containing aggregate operators should appear in the argument of the **having** clause
- "Find the departments where the average salary of employees working in office number 20 is higher than 25":

```
select Dept
from Employee
where Office = '20'
group by Dept
having avg(Salary) > 25
```

Syntax of an SQL Query …so far!

- Considering all clauses discussed so far, the syntax of an SQL query is:

`select` *TargetList*
`from` *TableList*
[**`where`** *Condition*]
[**`group by`** *GroupingAttributeList*]
[**`having`** *AggregateCondition*]
[**`order by`** *OrderingAttributeList*]

Set Queries

- A single select statement cannot represent any set operation.
- Syntax:

  SelectSQL { <**union** | **intersect** | **except** >
  [**all**] *SelectSQL* }

- "Find all first names and surnames of employees":

```
select FirstName as Name from Employee
 union
select Surname as Name from Employee
```

- Duplicates are removed (unless the **all** option is used)

Intersection

- "Find surnames of employees that are also first names":

```
select FirstName as Name
from Employee
 intersect
select Surname as Name
from Employee
```

(equivalent to:

```
select E1.FirstName as Name
from Employee E1, Employee E2
where E1.FirstName = E2.Surname )
```

Nested Queries

- The query appearing in the **where** clause is called a ***nested query***.

`select` *TargetList*
`from` *TableList*
[**`where`** *(select…)*]
[**`group by`** *GroupingAttributeList*]
[**`having`** *AggregateCondition*]
[**`order by`** *OrderingAttributeList*]

Nested Queries

- A **where** clause may include predicates that:

  1) Compare an attribute (or attribute expression) with the result of an SQL query;

  syntax: *ScalarValue Op* <**any** | **all**> *Select-Statement*

  any: *the predicate is true if at least one row returned by SelectSQL satisfies the comparison*

  all: predicate is true if all rows satisfy comparison;

  2) Use the existential quantifier on an SQL query;

  syntax: **exists** *Select-Statement*

  the predicate is true if *Select-Statement* is non-empty.

Simple Nested Query

- "Find the employees who work in departments in London":
  - without nested query:

```
select FirstName, Surname
from Employee, Department D
where Dept = DeptName and
      D.City = 'London'    )
```

  - with nested query:

```
select FirstName, Surname
  from Employee
  where Dept = any (select DeptName
      from Department
      where City = 'London')
```

…Another…

- "Find employees of the Planning department, having the same first name as a member of the Production department":
  - without nested query:

```
select E1.FirstName,E1.Surname
from Employee E1, Employee E2
where E1.FirstName=E2.FirstName and
        E2.Dept='Prod' and E1.Dept='Plan'
```

  - with a nested query:

```
select FirstName,Surname from Employee
where Dept = 'Plan' and FirstName = any
        (select FirstName from Employee
              where Dept = 'Prod')
```

Negation with Nested Queries

- "Find departments where there is no one named Brown":

  - Without a nested query:

```
select DeptName from Department
    except
select Dept from Employee
where Surname = 'Brown'
```

  - With a nested query:

```
select DeptName
from Department
where DeptName <>
      all (select Dept from Employee
          where Surname = 'Brown')
```

Operators in and not in

- Operator **in** is a shorthand for **= any**

```
select FirstName, Surname
from Employee
where Dept in (select DeptName
               from Department
               where City = 'London')
```

- Operator **not in** is a shorthand for **<> all**

```
select DeptName
from Department
where DeptName not in
   (select Dept from Employee
       where Surname = 'Brown')
```

max and min within a Nested Query

- Queries using the aggregate operators **max** and **min** can be expressed with nested queries
- "Find the department of the employee earning the highest salary":
  - with **max**:

```
select Dept from Employee
where Salary in (select max(Salary)
                 from Employee)
```

  - with a nested query:

```
select Dept from Employee
where Salary >= all (select Salary
                     from Employee)
```

Comments on Nested Queries

- The use of nested queries may produce less declarative queries, but often results in improved readability.
- Complex queries can become very difficult to understand.
- The use of variables must respect scoping conventions: a variable can be used only within the query where it is defined, or within a query that is recursively nested in the query where it is defined.