




# 3. User Interfaces and SQL Language (3/4)



# Some New Features of SQL

- CAST expression
- CASE expression
- **Sub-query**
- Outer Join
- Recursion



# Sub-query

- Embedded query & embedded query with correlation
- The functions of sub-queries have been enhanced in new SQL standard. Now they can be used in SELECT and FROM clause
  - Scalar sub-query
  - Table expression
  - Common table expression



# Scalar Sub-query

- The result of a sub-query is a single value. It can be used in the place where a value can occur.
- *Find the departments whose average bonus is higher than average salary :*

```
SELECT d.deptname, d.location
FROM dept AS d
WHERE (SELECT avg(bonus)
       FROM emp
       WHERE deptno=d.deptno)
> (SELECT avg(salary)
   FROM emp
   WHERE deptno=d.deptno)
```



## Scalar Sub-query

- *List the deptno, deptname, and the max salary of all departments located in New York :*

```
SELECT d.deptno, d.deptname, (SELECT MAX (salary)
                                FROM emp
                                WHERE deptno=d.deptno) AS maxpay
FROM dept AS d
WHERE d.location = 'New York' ;
```



# Table Expression

- The result of a sub-query is a table. It can be used in the place where a table can occur.

```
SELECT startyear, avg(pay)
FROM (SELECT name, salay+bonus AS pay,
             year(startdate) AS startyear
      FROM emp) AS emp2
GROUP BY startyear;
```

- *Find departments whose total payment is greater than 200000*

```
SELECT deptno, totalpay
FROM (SELECT deptno, sum(salay)+sum(bonus) AS totalpay
      FROM emp
      GROUP BY deptno) AS payroll
WHERE totalpay>200000;
```

- Table expressions are temporary views in fact.



# Common Table Expression

- In some complex query, a table expression may need occurring more than one time in the same SQL statements. Although it is permitted, the efficiency is low and there maybe inconsistency problem.
- WITH clause can be used to define a common table expression. In fact, it defines a temporary view.
- *Find the department who has the highest total payment :*



# Common Table Expression

- *Find the department who has the highest total payment :*

```
WITH payroll (deptno, totalpay) AS
    (SELECT deptno, sum(salary)+sum(bonus)
     FROM emp
     GROUP BY deptno)
SELECT deptno
FROM payroll
WHERE totalpay = (SELECT max(totalpay)
                  FROM payroll);
```

- Common table expression mainly used in queries which need multi level focuses.





# Common Table Expression

- *Find department pairs, in which the first department's average salary is more than two times of the second one's :*

```
WITH deptavg (deptno, avgsal) AS
    (SELECT deptno, avg(salary)
     FROM emp
     GROUP BY deptno)
SELECT d1.deptno, d1.avgsal, d2.deptno, d2.avgsal
FROM deptavg AS d1, deptavg AS d2
WHERE d1.avgsal > 2*d2.avgsal;
```



# Some New Features of SQL

- CAST expression
- CASE expression
- Sub-query
- **Outer Join**
- Recursion



# Outer Join

Teacher ( name, rank )

Course (subject, enrollment, quarter, teacher)

WITH

```
innerjoin(name, rank, subject, enrollment) AS
  (SELECT t.name, t.rank, c.subject, c.enrollment
   FROM teachers AS t, courses AS c
   WHERE t.name=c.teacher AND c.quarter='Fall 96') ,
```

```
teacher-only(name, rank) AS
```

```
(SELECT name, rank
 FROM teachers
 EXCEPT ALL
 SELECT name, rank
 FROM innerjoin) ,
```

```
course-only(subject, enrollment) AS
```

```
(SELECT subject, enrollment
 FROM courses
 EXCEPT ALL
 SELECT subject, enrollment
 FROM innerjoin)
```



# Outer Join

```
SELECT name, rank, subject, enrollment
FROM innerjoin
UNION ALL
SELECT name, rank,
       CAST (NULL AS Varchar(20)) AS subject,
       CAST (NULL AS Integer) AS enrollment
FROM teacher-only
UNION ALL
SELECT CAST (NULL AS Varchar(20)) AS name,
       CAST (NULL AS Varchar(20)) AS rank,
       subject, enrollment
FROM course-only ;
```



# Some New Features of SQL

- CAST expression
- CASE expression
- Sub-query
- Outer Join
- **Recursion**



# Recursion

- If a common table expression uses itself in its definition, this is called recursion. It can calculate a complex recursive inference in one SQL statement.  
**FedEmp** (name, salary, manager)
- *Find all employees under the management of Hoover and whose salary is more than 100000*

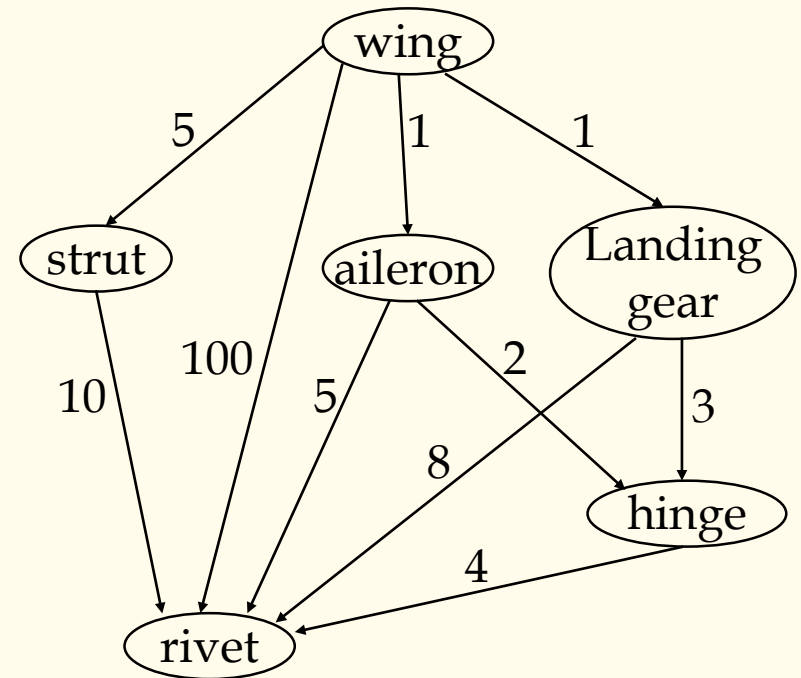
```
WITH agents (name, salary) AS
    ((SELECT name, salary                --- initial query
      FROM FedEmp
      WHERE manager='Hoover')
  UNION ALL
    (SELECT f.name, f.salary              --- recursive query
      FROM agents AS a, FedEmp AS f
      WHERE f.manager = a.name))
SELECT name                                --- final query
FROM agents
WHERE salary>100000 ;
```

# Recursive Calculation

- A classical “parts searching problem”

Components

Part	Subpart	QTY
wing	strut	5
wing	aileron	1
wing	landing gear	1
wing	rivet	100
strut	rivet	10
aileron	hinge	2
aileron	rivet	5
landing gear	hinge	3
landing gear	rivet	8
hinge	rivet	4



Directed acyclic graph, which assures the recursion can be stopped



# Recursive Calculation

- *Find how much rivets are used in one wing?*
- A temporary view is defined to show the list of each subpart's quantity used in a specified part :

WITH **wingpart** (subpart, qty) AS

((SELECT subpart, qty ---initial query

FROM components

WHERE part='wing')

UNION ALL

(SELECT c.subpart, w.qty\*c.qty ---recursive qry

FROM **wingpart** w, components c

WHERE w.subpart=c.part))

wingpart

Subpart	QTY	
strut	5	Used directly
aileron	1	Used directly
landing gear	1	Used directly
rivet	100	Used directly
rivet	50	Used on strut
hinge	2	Used on aileron
rivet	5	Used on aileron
hinge	3	on landing gear
rivet	8	on landing gear
rivet	8	on aileron hinges
rivet	12	on L G hinges





# Recursive Calculation

- *Find how much rivets are used in one wing?*

WITH wingpart (subpart, qty) AS

((SELECT subpart, qty ---initial query

FROM components

WHERE part='wing')

UNION ALL

(SELECT c.subpart, w.qty\*c.qty ---recursive qry

FROM wingpart w, components c

WHERE w.subpart=c.part))

SELECT sum(qty) AS qty

FROM wingpart

WHERE subpart='rivet' ;

- The result is :

qty
183



# Recursive Calculation

- *Find all subparts and their total quantity needed to assemble a wing :*

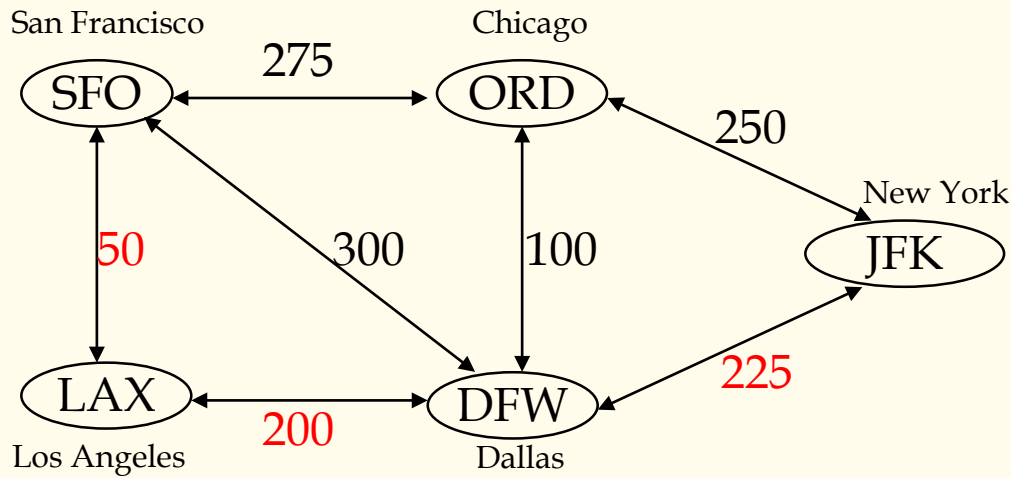
```
WITH wingpart (subpart, qty) AS
    ((SELECT subpart, qty          ---initial query
      FROM components
      WHERE part='wing')
  UNION ALL
    (SELECT c.subpart, w.qty*c.qty ---recursive qry
      FROM wingpart w, components c
      WHERE w.subpart=c.part))
SELECT subpart, sum(qty) AS qty
FROM wingpart
Group BY subpart ;
```

- The result is :

subpart	qty
strut	5
aileron	1
landing gear	1
hinge	5
rivet	183

# Recursive Search

- Typical airline route searching problem
- *Find the lowest total cost route from SFO to JFK*



Flights

FlightNo	Origin	Destination	Cost
HY 120	DFW	JFK	225
HY 130	DFW	LAX	200
HY 140	DFW	ORD	100
HY 150	DFW	SFO	300
HY 210	JFK	DFW	225
HY 240	JFK	ORD	250
HY 310	LAX	DFW	200
HY 350	LAX	SFO	50
HY 410	ORD	DFW	100
HY 420	ORD	JFK	250
HY 450	ORD	SFO	275
HY 510	SFO	DFW	300
HY 530	SFO	LAX	50
HY 540	SFO	ORD	275



# Recursive Search

```
WITH trips (destination, route, nsegs, totalcost) AS
  ((SELECT destination, CAST(destination AS varchar(20)), 1, cost
    FROM flights                                --- initial query
    WHERE origin='SFO')
  UNION ALL
  (SELECT f.destination,                                --- recursive query
    CAST(t.route || ',' || f.destination AS varchar(20)),
    t.nsegs+1, t.totalcost+f.cost
  FROM trips t, flights f
  WHERE t.destination=f.origin
    AND f.destination<>'SFO'                        --- stopping rule 1
    AND f.origin<>'JFK'                            --- stopping rule 2
    AND t.nsegs<=3))                               --- stopping rule 3
SELECT route, totalcost                            --- final query
FROM trips
WHERE destination='JFK' AND totalcost=              --- lowest cost rule
  (SELECT min(totalcost)
  FROM trips
  WHERE destination='JFK') ;
```



# Result

## Trips

Destination	Route	Nsegs	Totalcost
DFW	DFW	1	300
ORD	ORD	1	275
LAX	LAX	1	50
JFK	DFW, JFK	2	525
LAX	DFW, LAX	2	500
ORD	DFW, ORD	2	400
DFW	LAX, DFW	2	250
DFW	ORD, DFW	2	375
JFK	ORD, JFK	2	525
DFW	DFW, LAX, DFW	3	700
DFW	DFW, ORD, DFW	3	500
JFK	DFW, ORD, JFK	3	650
LAX	LAX, DFW, LAX	3	450
JFK	LAX, DFW, JFK	3	475
ORD	LAX, DFW, ORD	3	350
LAX	ORD, DFW, LAX	3	575
JFK	ORD, DFW, JFK	3	600
ORD	ORD, DFW, ORD	3	475

## Final result

route	totalcost
LAX, DFW, JFK	475



# Recursive Search

- *Only change the final query slightly, the least transfer time routes can be found :*

... ..

```
SELECT route, totalcost          --- final query
FROM trips
WHERE destination='JFK' AND nsegs=  --- least stop rule
      (SELECT min(nsegs)
       FROM trips
       WHERE destination='JFK') ;
```

Final result

route	totalcost
DFW, JFK	525
ORD, JFK	525