Introduction to Database Systems

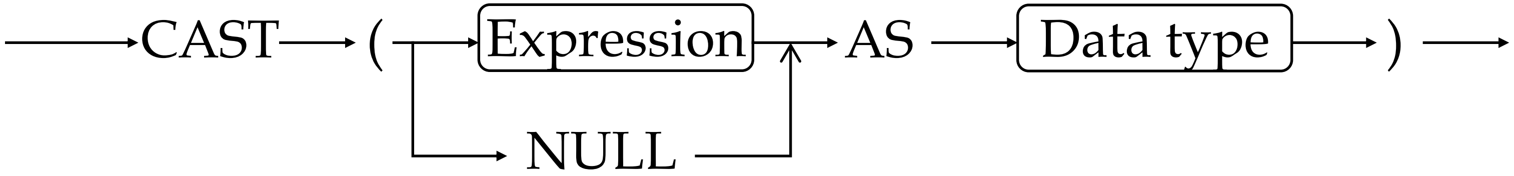
2023-Fall

3. User Interfaces and SQL Language

Some New Features of SQL

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

CAST Expression



- Change the expression to the target data type
- Valid target type
- Use
  - Match function parameters

    substr(string1, CAST(x AS Integer), CAST(y AS Integer))
  - Change precision while calculating

    CAST (elevation AS Decimal (5,0))
  - Assign a data type to NULL value

CAST Expression

- ***Example:***

  Students (name, school)

  Soldiers (name, service)

```
CREATE VIEW prospects (name, school, service) AS
        SELECT name, school, CAST(NULL AS Varchar(20))
        FROM Students
UNION
        SELECT name, CAST(NULL AS Varchar(20)), service
        FROM Soldiers ;
```

Some New Features of SQL

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

CASE Expression

- ***Simple form :***

  Officers (name, status, rank, title)

```
SELECT name, CASE status
                WHEN 1 THEN 'Active Duty'
                WHEN 2 THEN 'Reserve'
                WHEN 3 THEN 'Special Assignment'
                WHEN 4 THEN 'Retired'
                ELSE 'Unknown'
             END AS status
FROM Officers ;
```

CASE Expression

- ***General form (use searching condition):***

  Machines (serialno, type, year, hours_used, accidents)

- ***Find the rate of the accidents of "chain saw" in the whole accidents :***

```
SELECT sum (CASE
                WHEN type='chain saw' THEN accidents
                ELSE 0e0
           END) / sum (accidents)
FROM Machines;
```

CASE Expression

- ***Find the average accident rate of every kind of equipment :***
- **Machines (serialno, type, year, hours_used, accidents)**

```
SELECT type, CASE
                WHEN  sum(hours_used)>0 THEN
                      sum(accidents)/sum(hours_used)
                ELSE NULL
           END AS accident_rate
FROM Machines
GROUP BY type;
```

(Because some equipment maybe not in use at all, their hours_used is 0. Use CASE can prevent the expression divided by 0.)

CASE Expression

- ***Compared with***

```
SELECT type, sum(accidents)/sum(hours_used)
FROM Machines
GROUP BY type
HAVING sum(hours_used)>0;
```

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
    - result is a value
  - Table expression
    - result is a relation/table
  - Common table expression
    - a table appears in subquery multiple times…

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' names whose average bonus is higher than average salary:***
- **dept (deptno, deptname, location), emp (deptno, salary, bonus)**

```
SELECT d.deptname
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’ ;
```

- How about using **GROUP BY**?

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:***
  - *(Hint: we need to use paytoll table twice)*

```
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

"Inner" Joins: Another Syntax

```
SELECT s.*, r.bid
FROM Sailors s, Reserves r
WHERE s.sid = r.sid
AND ...
```

```
SELECT s.*, r.bid
FROM Sailors s INNER JOIN Reserves r
ON s.sid = r.sid
WHERE ...
```

Join Variants

SELECT <*column expression list*>
FROM *table_name*
[**INNER** | **NATURAL**
| {**LEFT** |**RIGHT** | **FULL** } {**OUTER**}] JOIN *table_name*
ON <*qualification_list*>
WHERE …

- INNER is default
- Inner join what we've learned so far
  - Same thing, just with different syntax.

Inner/Natural Joins

```
SELECT s.sid, s.sname, r.bid
FROM Sailors s, Reserves r
WHERE s.sid = r.sid
  AND s.age > 20;
```

SELECT s.sid, s.sname, r.bid
FROM Sailors s **NATURAL JOIN** Reserves r
WHERE s.age > 20;

SELECT s.sid, s.sname, r.bid
FROM Sailors s **INNER JOIN** Reserves r
ON s.sid = r.sid
WHERE s.age > 20;

- ***ALL 3 ARE EQUIVALENT!***
- “NATURAL” means equi-join for pairs of attributes with the same name

Example of Inner Join

Student Relation

ID	*name*	*dept_name*	*tot_cred*
00128	Zhang	Comp. Sci.	102
12345	Shankar	Comp. Sci.	32
19991	Brandt	History	80
23121	Chavez	Finance	110
44553	Peltier	Physics	56
45678	Levy	Physics	46
54321	Williams	Comp. Sci.	54
55739	Sanchez	Music	38
70557	Snow	Physics	0
76543	Brown	Comp. Sci.	58
76653	Aoi	Elec. Eng.	60
98765	Bourikas	Elec. Eng.	98
98988	Tanaka	Biology	120

Takes Relation

ID	*course_id*	*sec_id*	*semester*	*year*	*grade*
00128	CS-101	1	Fall	2017	A
00128	CS-347	1	Fall	2017	A-
12345	CS-101	1	Fall	2017	C
12345	CS-190	2	Spring	2017	A
12345	CS-315	1	Spring	2018	A
12345	CS-347	1	Fall	2017	A
19991	HIS-351	1	Spring	2018	B
23121	FIN-201	1	Spring	2018	C+
44553	PHY-101	1	Fall	2017	B-
45678	CS-101	1	Fall	2017	F
45678	CS-101	1	Spring	2018	B+
45678	CS-319	1	Spring	2018	B
54321	CS-101	1	Fall	2017	A-
54321	CS-190	2	Spring	2017	B+
55739	MU-199	1	Spring	2018	A-
76543	CS-101	1	Fall	2017	A
76543	CS-319	2	Spring	2018	A
76653	EE-181	1	Spring	2017	C
98765	CS-101	1	Fall	2017	C-
98765	CS-315	1	Spring	2018	B
98988	BIO-101	1	Summer	2017	A
98988	BIO-301	1	Summer	2018	*null*

Example of Inner Join

- **student natural join takes**

ID	name	dept_name	tot_cred	course_id	sec_id	semester	year	grade
00128	Zhang	Comp. Sci.	102	CS-101	1	Fall	2017	A
00128	Zhang	Comp. Sci.	102	CS-347	1	Fall	2017	A-
12345	Shankar	Comp. Sci.	32	CS-101	1	Fall	2017	C
12345	Shankar	Comp. Sci.	32	CS-190	2	Spring	2017	A
12345	Shankar	Comp. Sci.	32	CS-315	1	Spring	2018	A
12345	Shankar	Comp. Sci.	32	CS-347	1	Fall	2017	A
19991	Brandt	History	80	HIS-351	1	Spring	2018	B
23121	Chavez	Finance	110	FIN-201	1	Spring	2018	C+
44553	Peltier	Physics	56	PHY-101	1	Fall	2017	B-
45678	Levy	Physics	46	CS-101	1	Fall	2017	F
45678	Levy	Physics	46	CS-101	1	Spring	2018	B+
45678	Levy	Physics	46	CS-319	1	Spring	2018	B
54321	Williams	Comp. Sci.	54	CS-101	1	Fall	2017	A-
54321	Williams	Comp. Sci.	54	CS-190	2	Spring	2017	B+
55739	Sanchez	Music	38	MU-199	1	Spring	2018	A-
76543	Brown	Comp. Sci.	58	CS-101	1	Fall	2017	A
76543	Brown	Comp. Sci.	58	CS-319	2	Spring	2018	A
76653	Aoi	Elec. Eng.	60	EE-181	1	Spring	2017	C
98765	Bourikas	Elec. Eng.	98	CS-101	1	Fall	2017	C-
98765	Bourikas	Elec. Eng.	98	CS-315	1	Spring	2018	B
98988	Tanaka	Biology	120	BIO-101	1	Summer	2017	A
98988	Tanaka	Biology	120	BIO-301	1	Summer	2018	*null*

Course Relation

course_id	*title*	*dept_name*	*credits*
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

Dangerous in Natural Join

- Beware of ***unrelated attributes with same name*** which get equated incorrectly
- Example -- List the names of students instructors along with the titles of courses that they have taken
  - Correct version

    select *name*, *title*
    from *student* **natural join** *takes*, *course*
    where *takes*.*course_id* = *course*.*course_id*;
  - Incorrect version

    select *name*, *title*
    from *student* **natural join** *takes* **natural join** *course*;
    - This query omits all (student name, course title) pairs where the student takes a course in a department other than the student's own department.
    - The correct version (above), correctly outputs such pairs.

Outer Joins

- The extension of join operation. In join operation, only matching tuples fulfilling join conditions are left in results. Outer joins will keep unmated tuples, the vacant part is set ***Null*** :
  - Left outer join($* \bowtie$)

  Keep all tuples of left relation in the result.
  - Right outer join ($\bowtie *$)

  Keep all tuples of right relation in the result.
  - Full outer join ($* \bowtie *$)

  Keep all tuples of left and right relations in the result.

Outer Join Examples

- Relation *course*

course_id	*title*	*dept_name*	*credits*
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

- Relation *prereq*

course_id	*prereq_id*
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

- Observe that

  course information is missing CS-347

  prereq information is missing CS-315

Left Outer Join

- Returns all matched rows, and *preserves* all unmatched rows from the table on the left of the join clause
  - (use nulls in fields of non-matching tuples)

- *course* **natural left outer join** *prereq*
- In relational algebra: *course* $*\bowtie$ *prereq* ($⟕$)

course_id	*title*	*dept_name*	*credits*	*prereq_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	*null*

Right Outer Join

- Returns all matched rows, and *preserves* all unmatched rows from the table on the right of the join clause (use nulls in fields of non-matching tuples)
- *course* **natural right outer join** *prereq*
- In relational algebra: *course* ⋈* *prereq* (⟖)

course_id	*title*	*dept_name*	*credits*	*prereq_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-347	*null*	*null*	*null*	CS-101

Full Outer Join

- Returns all (matched or unmatched) rows from the tables on both sides of the join clause
- *course* **natural full outer join** *prereq*
- In relational algebra: *course* $*\bowtie*$ *prereq* ($⟗$)

course_id	*title*	*dept_name*	*credits*	*prereq_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	*null*
CS-347	*null*	*null*	*null*	CS-101

Full Outer Join

- <u>Returns all (matched or unmatched) rows from the tables on both sides</u> of the join clause

```
SELECT r.sid, b.bid, b.bname
FROM Reserves2 r FULL OUTER JOIN Boats2 b
ON r.bid = b.bid
```

- ***Returns all boats & all information on reservations***
- ***No match for r.bid? b.bid IS NULL AND b.bname IS NULL!***
- ***No match for b.bid? r.sid IS NULL!***

Joined Types and Conditions

- **Join operations** take two relations and return as a result another relation.
- These additional operations are typically used as subquery expressions in the **from** clause
- **Join type** – defines how tuples in each relation that do not match any tuple in the other relation (based on the join condition) are treated.
- **Join condition** – defines which tuples in the two relations match. *

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 19') ,
  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.
- ***Find all employees under the management of Hoover and whose salary is more than 100000***
- **FedEmp (name, salary, manager)**

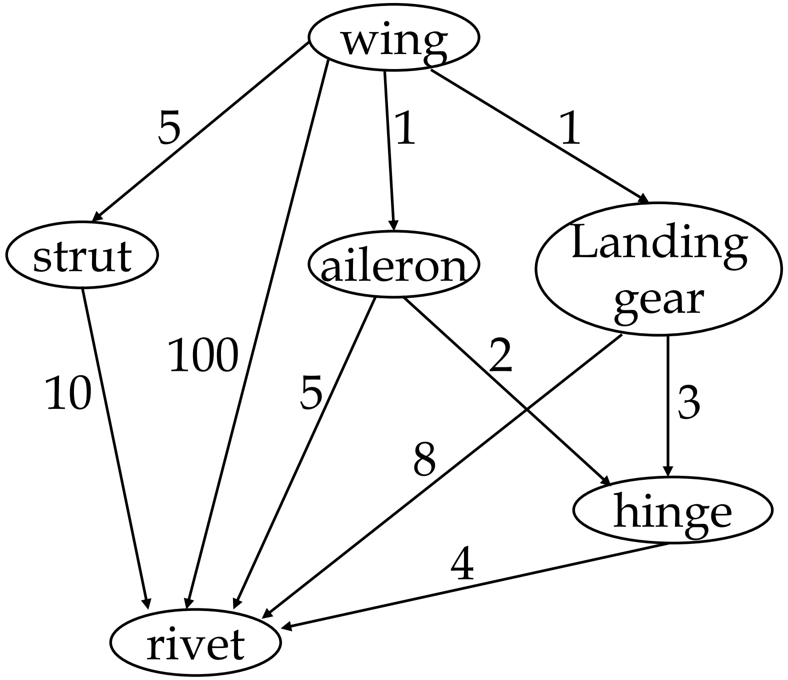
```
WITH agents (name, salary) AS
        ((SELECT name, salary                  --- initial query
          FROM FedEmp
          WHERE manager='Hoover')
        UNION ALL
         (SELECT f.name, f.salary              --- recursive query
          FROM agents AS a, FedEmp AS f
          WHERE f.manager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

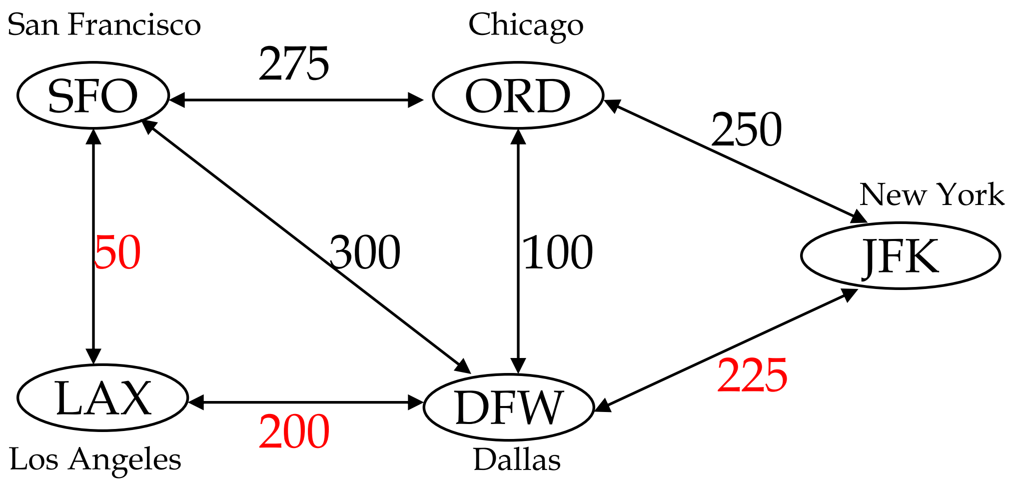
```
WITH wingpart (subpart, qty) AS
        ((SELECT subpart, qty                    ---initial query
          FROM components
          WHERE part='wing')
        UNION ALL
         (SELECT c.subpart, w.qty*c.qty    ---recursive qry
          FROM wingpart w, components c
          WHERE w.subpart=c.part))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

Views

- In some cases, it is not desirable for all users to see the entire logical model (that is, all the actual relations stored in the database.)
- Consider a person who needs to know an instructors name and department, but not the salary. This person should see a relation described, in SQL, by

  select *ID*, *name*, *dept_name*
  from *instructor*

- A **view** provides a mechanism to hide certain data from the view of certain users.
- Any relation that is not of the conceptual model but is made visible to a user as a "virtual relation" is called a **view**.

Views: Named Queries

CREATE VIEW *view_name*
AS *select_statement*

- Makes development simpler
- Often used for security
- Not “materialized”

- Once a view is defined, the view name can be used to refer to the virtual relation that the view generates.
- View definition is not the same as creating a new relation by evaluating the query expression
  - Rather, a view definition causes the saving of an expression; the expression is substituted into queries using the view.

View Definition and Use

- A view of instructors without their salary

```
create view faculty as
    select ID, name, dept_name
    from instructor
```

- Find all instructors in the Biology department

```
select name
from faculty
where dept_name = 'Biology'
```

- Create a view of department salary totals

```
create view departments_total_salary(dept_name, total_salary) as
    select dept_name, sum (salary)
    from instructor
    group by dept_name;
```

Views Instead of Relations in Queries

```
CREATE VIEW Redcount
AS SELECT B.bid, COUNT(*) AS scount
        FROM Boats2 B, Reserves2 R
        WHERE R.bid=B.bid AND B.color='red'
        GROUP BY B.bid;
```

```
SELECT * from redcount;
```

bid	scount
102	1

```
SELECT bname, scount
FROM Redcount R, Boats2 B
WHERE R.bid=B.bid
AND scount < 10;
```

Views Defined Using Other Views

- One view may be used in the expression defining another view
- A view relation v_1 is said to ***depend directly*** on a view relation v_2 if v_2 is used in the expression defining v_1
- A view relation v_1 is said to ***depend on*** view relation v_2 if either v_1 depends directly to v_2 or there is a path of dependencies from v_1 to v_2
- A view relation v is said to be ***recursive*** if it depends on itself.

Views Defined Using Other Views

- **create view** *physics_fall_2017* **as**
  select *course.course_id, sec_id, building, room_number*
  from *course, section*
  where *course.course_id = section.course_id*
  and *course.dept_name* = 'Physics'
  and *section.semester* = 'Fall'
  and *section.year* = '2017';

- **create view** ***physics_fall_2017_watson*** **as**
  select *course_id, room_number*
  from ***physics_fall_2017***
  where *building*= 'Watson';

Materialized Views

- Certain database systems allow view relations to be physically stored.
  - Physical copy created when the view is defined.
  - Such views are called **Materialized view**:
- If relations used in the query are updated, the materialized view result becomes out of date
  - Need to **maintain** the view, by updating the view whenever the underlying relations are updated.

Update of a View

- Add a new tuple to *faculty* view which we defined earlier

    insert into *faculty*

        values ('30765', 'Green', 'Music');

- This insertion must be represented by the insertion into the ***instructor*** relation
  - Must have a value for salary.
  - Two approaches
    - Reject the insert
    - Insert the tuple ('30765', 'Green', 'Music', null) into the *instructor* relation

Some Updates Cannot be Translated Uniquely

- **create view** *instructor_info* **as**
  select *ID, name, building*
  from *instructor, department*
  where *instructor.dept_name* = *department.dept_name*;
- **insert into** *instructor_info*
  values ('69987', 'White', 'Taylor');
- Issues
  - Which department, if multiple departments in Taylor?
  - What if no department is in Taylor?

View Updates in SQL

- Most SQL implementations allow updates only on simple views
  - The **from** clause has only one database relation.
  - The **select** clause contains only attribute names of the relation, and does not have any expressions, aggregates, or **distinct** specification.
  - Any attribute not listed in the **select** clause can be set to null.
  - The query does not have a **group** by or **having** clause.

Summary

- You've now seen SQL—you are armed.
- A declarative language
  - Somebody has to translate to algorithms though…
  - The RDBMS implementor ... i.e. you!
- The data structures and algorithms that make SQL possible also power:
  - NoSQL, data mining, scalable ML, network routing…
  - A toolbox for scalable computing!