CS4.301 Data & Applications

Ponnurangam Kumaraguru (“PK”)
#ProfGiri @ IIIT Hyderabad

 pk.profgiri  /in/ponguru  @ponguru  Ponnurangam.kumaraguru

Ambiguous Attribute Names

Same name can be used for two (or more) attributes in different relations

- As long as the attributes are in different relations
- Must **qualify** the attribute name with the relation name to prevent ambiguity

```
Q1A:    SELECT    Fname, EMPLOYEE.Name, Address
        FROM      EMPLOYEE, DEPARTMENT
        WHERE     DEPARTMENT.Name='Research' AND
                  DEPARTMENT.Dnumber=EMPLOYEE.Dnumber;
```

Aliasing, and Renaming

Aliases or **tuple variables**

Declare alternative relation names E and S to refer to the EMPLOYEE relation twice in a query:

Query 8. For each employee, retrieve the employee's first and last name and the first and last name of his or her immediate supervisor.

```
SELECT  E.Fname, E.Lname, S.Fname, S.Lname
FROM    EMPLOYEE AS E, EMPLOYEE AS S
WHERE E.Super_ssn=S.Ssn;
```

Recommended practice to abbreviate names and to prefix same or similar attribute from multiple tables.

Aliasing,Renaming and Tuple Variables (contd.)

The attribute names can also be renamed

```
EMPLOYEE AS E(Fn, Mi, Ln, Ssn, Bd, Addr, Sex,
Sal, Sssn, Dno)
```

Note that the relation EMPLOYEE now has a variable name E which corresponds to a tuple variable

The “AS” may be dropped in most SQL implementations

Nested Queries, Tuples, and Set/Multiset Comparisons

Nested queries

- Complete select-from-where blocks within WHERE clause of another query
- **Outer query and nested subqueries**

Comparison operator `IN`

- Compares value *v* with a set (or multiset) of values *V*
- Evaluates to `TRUE` if *v* is one of the elements in *V*

Nested Queries (cont'd.)

```
SELECT   DISTINCT Pnumber
FROM     PROJECT
WHERE    Pnumber IN
         (SELECT   Pnumber
         FROM      PROJECT, DEPARTMENT, EMPLOYEE
         WHERE     Dnum = Dnumber AND
         Mgr_ssn = Ssn and Lname = 'Smith')
         OR
         Pnumber IN
         (SELECT   Pno
         FROM      WORKS_ON, EMPLOYEE
         WHERE     Essn = Ssn AND Lname = 'Smith');
```

```
+---------+
| Pnumber |
+---------+
|       1 |
|       2 |
+---------+
2 rows in set (0.01 sec)
```

Nested Queries (cont'd.)

```
SELECT      DISTINCT    Pnumber
FROM        PROJECT
WHERE       Pnumber IN
            (SELECT     Pnumber
            FROM        PROJECT, DEPARTMENT, EMPLOYEE
            WHERE       Dnum = Dnumber AND
            Mgr_ssn = Ssn and Lname = 'Smith')
            OR
            Pnumber IN
            (SELECT     Pno
            FROM        WORKS_ON, EMPLOYEE
            WHERE       Essn = Ssn AND Lname = 'Smith');
```

```
mysql> (SELECT DISTINCT Pnumber
    -> FROM PROJECT, DEPARTMENT, EMPLOYEE
    -> WHERE Dnum=Dnumber AND Mgr_ssn=Ssn
    -> AND Lname='Smith')
    -> UNION
    -> (SELECT DISTINCT Pnumber
    -> FROM PROJECT, WORKS_ON, EMPLOYEE
    -> WHERE Pnumber=Pno AND Essn=Ssn
    -> AND Lname='Smith');
+---------+
| Pnumber |
+---------+
|       1 |
|       2 |
+---------+
2 rows in set (0.00 sec)
```

Nested Queries (cont'd.)

Use tuples of values in comparisons

Place them within parentheses

```
Select distinct essn
From works_on
Where (pno, hours) IN
(Select pno, hours from
works_on where essn =
'123456789');
```

```
mysql> Select pno, hours from works_on where essn = '123456789';
+-----+-------+
| pno | hours |
+-----+-------+
|   1 |  32.5 |
|   2 |   7.5 |
+-----+-------+
2 rows in set (0.04 sec)
```

```
mysql> Select distinct essn
    -> From works_on
    -> Where (pno, hours) IN (Select pno, hours from works_on where essn = '123456789');
+-----------+
| essn      |
+-----------+
| 123456789 |
+-----------+
1 row in set (0.01 sec)
```

Nested Queries (cont'd.)

Use other comparison operators to compare a single value *v*

- `= ANY` (or `= SOME`) operator
  - Returns `TRUE` if the value *v* is equal to some value in the set *V* and is hence equivalent to `IN`
- Other operators that can be combined with `ANY` (or `SOME`): >, >=, <, <=, and <>
- `ALL`: value must exceed all values from nested query

```
mysql> select salary from employee where dno = 5;
+--------+
| salary |
+--------+
|  30000 |
|  40000 |
|  25000 |
|  38000 |
+--------+
4 rows in set (0.00 sec)
```

Select lname, fname, salary from employee where salary > all (select salary from employee where dno = 5);

```
mysql> Select lname, fname, salary from employee w
here salary > all (select salary from employee whe
re dno = 5);
+---------+----------+--------+
| lname   | fname    | salary |
+---------+----------+--------+
| Borg    | James    |  55000 |
| Wallace | Jennifer |  43000 |
+---------+----------+--------+
2 rows in set (0.00 sec)
```

Correlated Nested Queries

Queries that are nested using the = or IN comparison operator can be collapsed into one single block, last query can be changed like

```
Select e.fname, e.lname from
employee as e where e.ssn in
(select essn from dependent as
d where
e.fname=d.dependent_name
and e.sex=d.sex);
```

```
SELECT E.Fname, E.Lname
FROM EMPLOYEE AS E,
DEPENDENT AS D WHERE
E.Ssn=D.Essn AND E.Sex=D.Sex
AND
E.Fname=D.Dependent_name;
```

Correlated nested query

Evaluated once for each tuple in the outer query

Explicit Sets and Renaming of Attributes in SQL

Can use explicit set of values in WHERE clause

SELECT DISTINCT Essn FROM WORKS_ON WHERE Pno IN (1, 2, 3);

```
mysql> SELECT DISTINCT Essn FROM WORKS_ON WHERE Pno
 IN (1, 2, 3);
+-----------+
| Essn      |
+-----------+
| 123456789 |
| 453453453 |
| 333445555 |
| 666884444 |
+-----------+
4 rows in set (0.00 sec)
```

Explicit Sets and Renaming of Attributes in SQL

Use qualifier AS followed by desired new name

Rename any attribute that appears in the result of a query

Select e.lname as employee_name, s.lname as supervisor_name from employee as e, employee as s where e.super_ssn = s.ssn;

```
mysql> Select e.lname as employee_name, s.lname as
supervisor_name from employee as e, employee as s w
here e.super_ssn = s.ssn;
+---------------+-----------------+
| employee_name | supervisor_name |
+---------------+-----------------+
| Smith         | Wong            |
| Wong          | Borg            |
| English       | Wong            |
| Narayan       | Wong            |
| Wallace       | Borg            |
| Jabbar        | Wallace         |
| Zelaya        | Wallace         |
+---------------+-----------------+
7 rows in set (0.00 sec)
```

Renaming Results of Aggregation

```
SELECT SUM(Salary) AS
Total_Sal, MAX(Salary)
AS Highest_Sal,
MIN(Salary) AS
Lowest_Sal, AVG(Salary)
AS Average_Sal FROM
EMPLOYEE;
```

```
mysql> SELECT SUM(Salary) AS Total_Sal, MAX(Salary) AS Highe
st_Sal, MIN(Salary) AS Lowest_Sal, AVG(Salary) AS Average_Sa
l FROM EMPLOYEE;
+-----------+-------------+------------+-------------+
| Total_Sal | Highest_Sal | Lowest_Sal | Average_Sal |
+-----------+-------------+------------+-------------+
|    281000 |       55000 |      25000 |  35125.0000 |
+-----------+-------------+------------+-------------+
1 row in set (0.00 sec)
```

Aggregate Functions in SQL (cont'd.)

Queries 21 and 22. Retrieve the total number of employees in the company (Q21) and the number of employees in the 'Research' department (Q22).

```
Select count(*) from
employee;

Select count(*) from
employee, department
where dno=dnumber
and dname='research';
```

```
mysql> Select count(*) from employee;
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.02 sec)

mysql> Select count(*) from employee, department where dno=d
number and dname='research';
+----------+
| count(*) |
+----------+
|        4 |
+----------+
1 row in set (0.00 sec)
```

Group BY example

```
SELECT Dno, COUNT(*),
AVG(Salary) FROM
EMPLOYEE GROUP BY
Dno;
```

```
mysql> SELECT Dno, COUNT(*), AVG(Salary) FROM EMPLOYEE GROUP
 BY Dno;
+-----+----------+-------------+
| Dno | COUNT(*) | AVG(Salary) |
+-----+----------+-------------+
|   5 |        4 |  33250.0000 |
|   1 |        1 |  55000.0000 |
|   4 |        3 |  31000.0000 |
+-----+----------+-------------+
3 rows in set (0.01 sec)
```

Group BY example

```
SELECT Pnumber,
Pname, COUNT(*) FROM
PROJECT, WORKS_ON
WHERE Pnumber=Pno
GROUP BY Pname;
```

```
mysql> SELECT Pnumber, Pname, COUNT(*) FROM PROJECT, WORKS_O
N WHERE Pnumber=Pno GROUP BY Pname;
+---------+-----------------+----------+
| Pnumber | Pname           | COUNT(*) |
+---------+-----------------+----------+
|      10 | Computerization |        3 |
|      30 | Newbenefits     |        3 |
|       1 | ProductX        |        2 |
|       2 | ProductY        |        3 |
|       3 | ProductZ        |        2 |
|      20 | Reorganization  |        3 |
+---------+-----------------+----------+
6 rows in set (0.01 sec)
```

Grouping: The GROUP BY and HAVING Clauses (cont'd.)

HAVING clause

Provides a condition to select or reject an entire group:

Query 26. For each project *on which more than two employees work,* retrieve the project number, the project name, and the number of employees who work on the project.

```
SELECT Pnumber,
Pname, COUNT(*) FROM
PROJECT, WORKS_ON
WHERE Pnumber=Pno
GROUP BY Pnumber
HAVING COUNT(*) > 2;
```

```
mysql> SELECT Pnumber, Pname, COUNT(*) FROM PROJECT, WORKS_O
N WHERE Pnumber=Pno GROUP BY Pnumber HAVING COUNT(*) > 2;
+---------+-----------------+----------+
| Pnumber | Pname           | COUNT(*) |
+---------+-----------------+----------+
|       2 | ProductY        |        3 |
|      10 | Computerization |        3 |
|      20 | Reorganization  |        3 |
|      30 | Newbenefits     |        3 |
+---------+-----------------+----------+
4 rows in set (0.00 sec)
```

EXPANDED Block Structure of SQL Queries

SELECT <attribute and function list>
FROM <table list>
[**WHERE** <condition>]
[**GROUP BY** <grouping attribute(s)>]
[**HAVING** <group condition>]
[**ORDER BY** <attribute list>];

Specification of Views in SQL

`CREATE VIEW` command

Give table name, list of attribute names, and a query to specify the contents of the view

Create view works_on1 as select fname, lname, hours from employee, project, works_on where ssn=essn and pno=pnumber;

```
mysql> Create view works_on1 as select fname, lname, hours f
rom employee, project, works_on where ssn=essn and pno=pnumb
er;
Query OK, 0 rows affected (0.05 sec)
```

Data in view, query view

```
select * from works_on1;
```

```
mysql> select * from works_on1;
+----------+---------+-------+
| fname    | lname   | hours |
+----------+---------+-------+
| Franklin | Wong    |  10.0 |
| James    | Borg    |  16.0 |
| Jennifer | Wallace |  15.0 |
| Franklin | Wong    |  10.0 |
| Ahmad    | Jabbar  |  35.0 |
| Alicia   | Zelaya  |  10.0 |
| Jennifer | Wallace |  20.0 |
| Ahmad    | Jabbar  |   5.0 |
| Alicia   | Zelaya  |  30.0 |
| John     | Smith   |  32.5 |
| Joyce    | English |  20.0 |
| John     | Smith   |   7.5 |
| Franklin | Wong    |  10.0 |
| Joyce    | English |  20.0 |
| Franklin | Wong    |  10.0 |
| Ramesh   | Narayan |  40.0 |
+----------+---------+-------+
16 rows in set (0.01 sec)
```

select fname, lname from works_on1 where hours=10;

```
mysql> select fname, lname from works_on1 where hours=10;
+----------+--------+
| fname    | lname  |
+----------+--------+
| Franklin | Wong   |
| Franklin | Wong   |
| Franklin | Wong   |
| Franklin | Wong   |
| Alicia   | Zelaya |
+----------+--------+
5 rows in set (0.01 sec)
```

The DROP Command

`DROP` command

Used to drop named schema elements, such as tables, domains, or constraint

Drop behavior options:

`CASCADE` and `RESTRICT`

RESTRICT – schema will be dropped only if it has no elements in it

Example:

```
DROP SCHEMA COMPANY CASCADE;
```

This removes the schema and all its elements including tables, views, constraints, etc.

```
DROP TABLE DEPENDENT CASCADE;
```

If we no longer wish to track the dependents

In-Class Activity

- Schema & Data: https://github.com/tolgahanakgun/Elmasri-Database

a. Retrieve the names of all employees who work in the department that has the employee with the highest salary among all employees.

b. Retrieve the names of all employees whose supervisor's supervisor has ''888665555' for ssn

c. Retrieve the names of employees who make at least 10,000 USD more than the employee who is paid the least in the company

d. Create a view that has the department name, manager name, and manager salary for every department

e. Create a view that has the employee name, supervisor name, and employee salary for each employee who works in the 'research' department

This lecture

The ALTER table command

Alter table actions include:

- Adding or dropping a column (attribute)
- Changing a column definition
- Adding or dropping table constraints

Example:

```
ALTER TABLE COMPANY.EMPLOYEE ADD COLUMN Job
VARCHAR(12);
```

Keeping track of jobs of employees

Adding and Dropping Constraints

Change constraints specified on a table

- Add or drop a named constraint

```
ALTER TABLE COMPANY.EMPLOYEE
DROP CONSTRAINT EMPSUPERFK CASCADE;
```

To be dropped, a constraint must have been given a name when it is specified

Dropping Columns, Default Values

To drop a column

Choose either `CASCADE` or `RESTRICT`

`CASCADE` would drop the column from views etc. `RESTRICT` is possible if no views refer to it.

ALTER TABLE COMPANY.EMPLOYEE **DROP COLUMN** Address **CASCADE**;

removes the attribute Address from the employee base table

Default values can be dropped and altered :

ALTER TABLE COMPANY.DEPARTMENT **ALTER COLUMN** Mgr_ssn **DROP DEFAULT**;

ALTER TABLE COMPANY.DEPARTMENT **ALTER COLUMN** Mgr_ssn **SET DEFAULT** '333445555';

The EXISTS and UNIQUE Functions in SQL for correlating queries

`EXISTS` function

Check whether the result of a correlated nested query is empty or not. They are Boolean functions that return a TRUE or FALSE result.

`EXISTS` and `NOT EXISTS`

Typically used in conjunction with a correlated nested query

SQL function `UNIQUE(Q)`

Returns `TRUE` if there are no duplicate tuples in the result of query Q

USE of EXISTS

SELECT Fname, Lname FROM Employee WHERE EXISTS (SELECT * FROM DEPENDENT WHERE Ssn= Essn);

```
mysql> SELECT Fname, Lname FROM Employee WHERE EXIS
TS (SELECT *
    -> FROM DEPENDENT WHERE Ssn= Essn);
+----------+---------+
| Fname    | Lname   |
+----------+---------+
| John     | Smith   |
| Franklin | Wong    |
| Jennifer | Wallace |
+----------+---------+
3 rows in set (0.00 sec)
```

USE OF NOT EXISTS

SELECT Fname, Lname FROM Employee WHERE NOT EXISTS (SELECT Pnumber FROM PROJECT WHERE Dno=5);

```
mysql> SELECT Fname, Lname FROM Employee WHERE NOT
EXISTS (SELECT  Pnumber FROM PROJECT WHERE Dno=5);
+----------+---------+
| Fname    | Lname   |
+----------+---------+
| James    | Borg    |
| Jennifer | Wallace |
| Ahmad    | Jabbar  |
| Alicia   | Zelaya  |
+----------+---------+
4 rows in set (0.01 sec)
```

Specifying Joined Tables in the FROM Clause of SQL

Joined table

Permits users to specify a table resulting from a join operation in the FROM clause of a query

The FROM clause in Q1A

Contains a single joined table. JOIN may also be called INNER JOIN

Select fname, lname, address
from (employee join department
on dno=dnumber) where
dname='research';

```
mysql> Select fname, lname, address from (employee
join department on dno=dnumber) where dname='resear
ch';
+----------+---------+-------------------------+
| fname    | lname   | address                 |
+----------+---------+-------------------------+
| John     | Smith   | 731 Fondren, Houston TX |
| Franklin | Wong    | 638 Voss, Houston TX    |
| Joyce    | English | 5631 Rice, Houston TX   |
| Ramesh   | Narayan | 975 Fire Oak, Humble TX |
+----------+---------+-------------------------+
4 rows in set (0.04 sec)
```

Different Types of JOINed Tables in SQL

Specify different types of join

- NATURAL JOIN
- Various types of OUTER JOIN (LEFT, RIGHT, FULL)

NATURAL JOIN on two relations R and S

- No join condition specified
- Is equivalent to an implicit EQUIJOIN condition for each pair of attributes with same name from R and S
- The associated tables have one or more pairs of identically named columns
- The columns must be the same data type
- No need for ON

NATURAL JOIN

```
mysql> select Fname, Lname, Address FROM (EMPLOYEE NATURAL JOIN DEPARTMEN
T) WHERE Dname='Research';
+----------+---------+-------------------------+
| Fname    | Lname   | Address                 |
+----------+---------+-------------------------+
| John     | Smith   | 731 Fondren, Houston TX |
| Franklin | Wong    | 638 Voss, Houston TX    |
| Joyce    | English | 5631 Rice, Houston TX   |
| Ramesh   | Narayan | 975 Fire Oak, Humble TX |
| James    | Borg    | 450 Stone, Houston TX   |
| Jennifer | Wallace | 291 Berry, Bellaire TX  |
| Ahmad    | Jabbar  | 980 Dallas, Houston TX  |
| Alicia   | Zelaya  | 3321 Castle, Spring TX  |
+----------+---------+-------------------------+
8 rows in set (0.01 sec)
```

INNER and OUTER Joins

INNER JOIN **(versus** OUTER JOIN**)**

- Default type of join in a joined table
- Tuple is included in the result only if a matching tuple exists in the other relation

LEFT OUTER JOIN

- Every tuple in left table must appear in result
- If no matching tuple
  - Padded with NULL values for attributes of right table

RIGHT OUTER JOIN

- Every tuple in right table must appear in result
- If no matching tuple
  - Padded with NULL values for attributes of left table

Natural join & Inner join difference

Comparing Natural Join and Inner Join in SQL

S.No.	NATURAL JOIN	INNER JOIN
1.	Natural join is a join operation that merges two tables based on matching column names and data types.	Inner join operates with a specific join condition, forming a new table by pairing column values of two tables according to the join-predicate.
2.	The final table resulting from a natural join will contain all the attributes of both the tables without duplicating the column.	The final table resulting from an inner join will contain all the attributes of both the tables, including duplicate columns.
3.	Natural joins are not supported by SQL Server Management Studio.	SQL Server Management Studio fully supports inner joins.

```
SELECT *
FROM company
INNER JOIN foods
ON company.company_id = foods.company_id;
```

Output:

```
COMPANY_ID COMPANY_NAME    COMPANY_CITY    ITEM_ID    ITEM_NAME       ITEM_UNIT  COMPANY_ID
---------- --------------- --------------- ---------- --------------- ---------- ----------
16         Akas Foods      Delhi           1          Chex Mix        Pcs        16
15         Jack Hill Ltd   London          6          Cheez-It        Pcs        15
15         Jack Hill Ltd   London          2          BN Biscuit      Pcs        15
17         Foodies.        London          3          Mighty Munch    Pcs        17
15         Jack Hill Ltd   London          4          Pot Rice        Pcs        15
18         Order All       Boston          5          Jaffa Cakes     Pcs        18
```

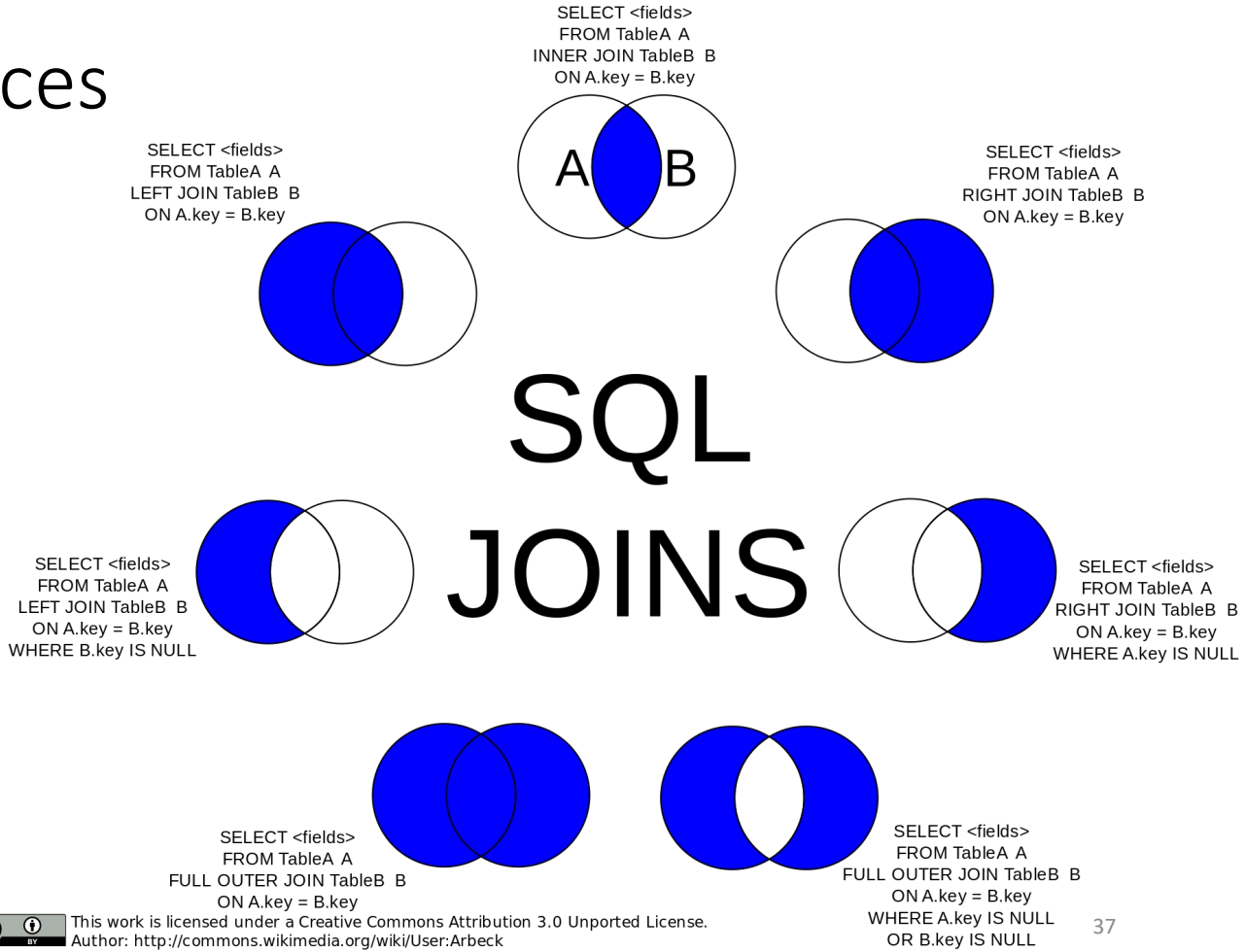
```
SELECT *
FROM company
NATURAL JOIN foods;
```

Copy

Output:

```
COMPANY_ID COMPANY_NAME    COMPANY_CITY    ITEM_ID    ITEM_NAME       ITEM_UNIT
---------- --------------- --------------- ---------- --------------- ----------
16         Akas Foods      Delhi           1          Chex Mix        Pcs
15         Jack Hill Ltd   London          6          Cheez-It        Pcs
15         Jack Hill Ltd   London          2          BN Biscuit      Pcs
17         Foodies.        London          3          Mighty Munch    Pcs
15         Jack Hill Ltd   London          4          Pot Rice        Pcs
18         Order All       Boston          5          Jaffa Cakes     Pcs
```

Joins differences



https://i.stack.imgur.com/3bs7C.png

Multiway JOIN in the FROM clause

FULL OUTER JOIN – combines result if LEFT and RIGHT OUTER JOIN

Can nest JOIN specifications for a multiway join:

```
SELECT Pnumber, Dnum, Lname,
Address, Bdate FROM ((PROJECT
JOIN DEPARTMENT ON
Dnum=Dnumber) JOIN EMPLOYEE
ON Mgr_ssn=Ssn) WHERE
Plocation='Stafford';
```

```
mysql> SELECT Pnumber, Dnum, Lname, Address, Bdate FROM ((PROJECT JOIN DE
PARTMENT ON Dnum=Dnumber)  JOIN EMPLOYEE ON Mgr_ssn=Ssn) WHERE   Plocatio
n='Stafford';
+---------+------+---------+-------------------------+------------+
| Pnumber | Dnum | Lname   | Address                 | Bdate      |
+---------+------+---------+-------------------------+------------+
|      10 |    4 | Wallace | 291 Berry, Bellaire TX | 1941-06-20 |
|      30 |    4 | Wallace | 291 Berry, Bellaire TX | 1941-06-20 |
+---------+------+---------+-------------------------+------------+
2 rows in set (0.02 sec)
```

CHAPTER 14

Basics of Functional Dependencies and Normalization for Relational Databases

Informal Design Guidelines for Relational Databases (2)

We first discuss informal guidelines for good relational design

Then we discuss formal concepts of functional dependencies and normal forms

- 1NF (First Normal Form)
- 2NF (Second Normal Form)
- 3NF (Third Noferferferfewrmal Form)
- BCNF (Boyce-Codd Normal Form)

Additional types of dependencies, further normal forms, relational design algorithms by synthesis are discussed in Chapter 15

1.1 Semantics of the Relational Attributes must be clear

GUIDELINE 1: Informally, each tuple in a relation should represent one entity or relationship instance. (Applies to individual relations and their attributes).

- Attributes of different entities (EMPLOYEEs, DEPARTMENTs, PROJECTs) should not be mixed in the same relation
- Only foreign keys should be used to refer to other entities

Bottom Line: *Design a schema that can be explained easily relation by relation. The semantics of attributes should be easy to interpret.*

Figure 14.1 A simplified COMPANY relational database schema

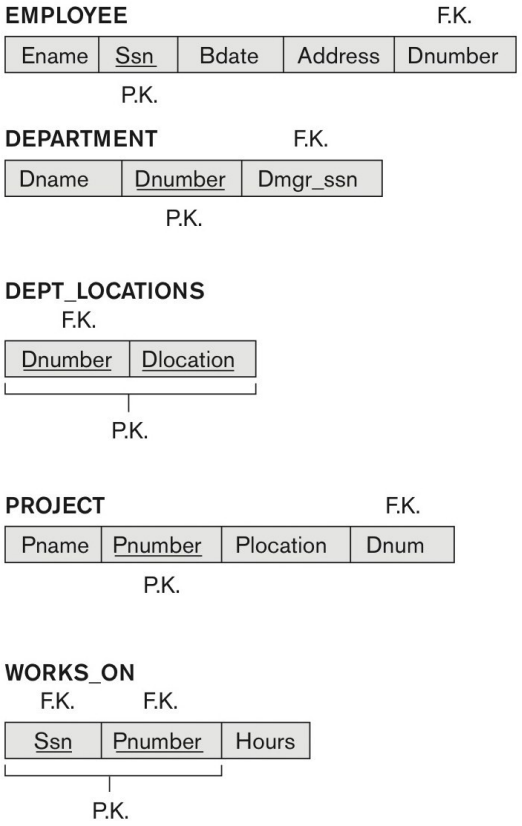
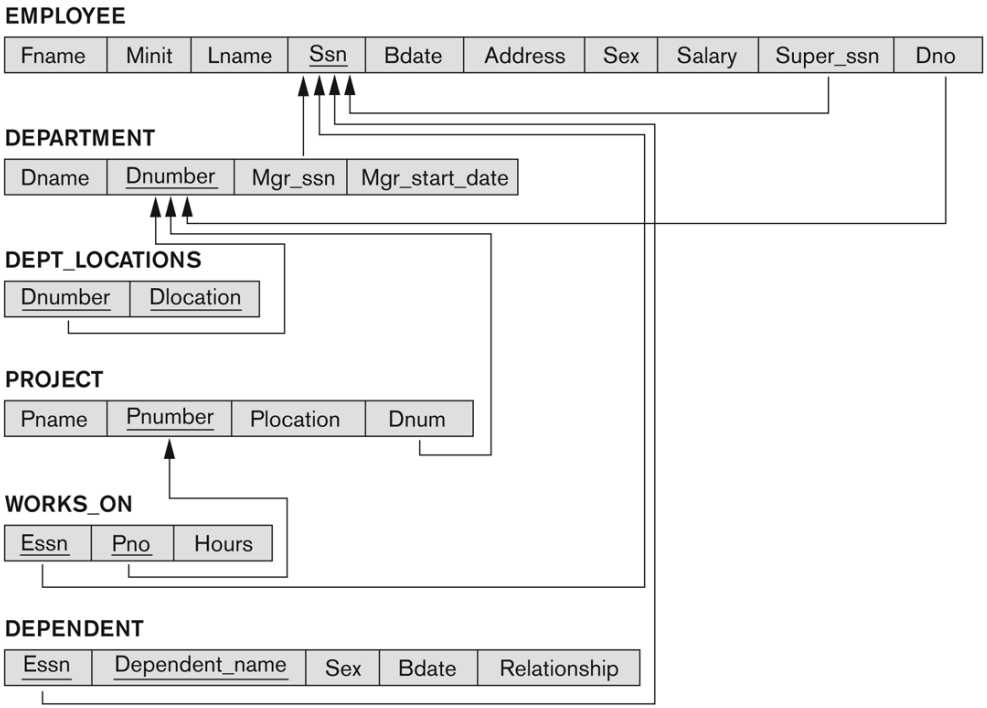
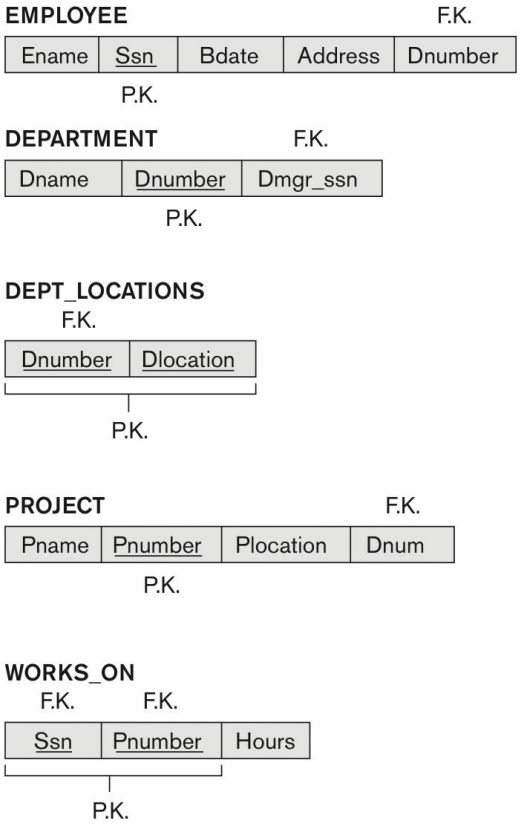


Figure 14.1 A simplified COMPANY relational database schema.

Figure 14.1 A simplified COMPANY relational database schema

Figure 5.7
Referential integrity constraints displayed on the COMPANY relational database schema.





EMPLOYEE

Ename	Ssn	Bdate	Address	Dnumber
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4
Wallace, Jennifer S.	987654321	1941-06-20	291Berry, Bellaire, TX	4
Narayan, Ramesh K.	666884444	1962-09-15	975 Fire Oak, Humble, TX	5
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX	5
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX	4
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX	1

DEPARTMENT

Dname	Dnumber	Dmgr_ssn
Research	5	333445555
Administration	4	987654321
Headquarters	1	888665555

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

WORKS_ON

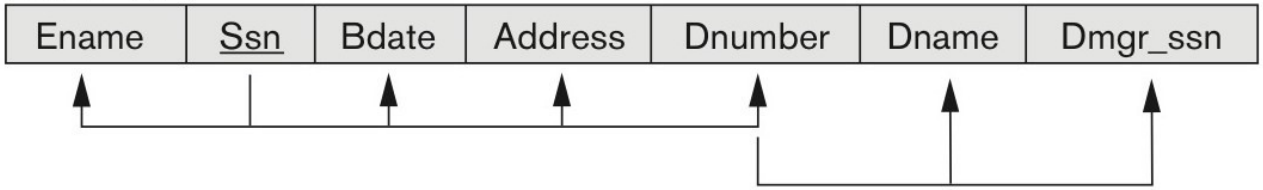
Ssn	Pnumber	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	Null

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

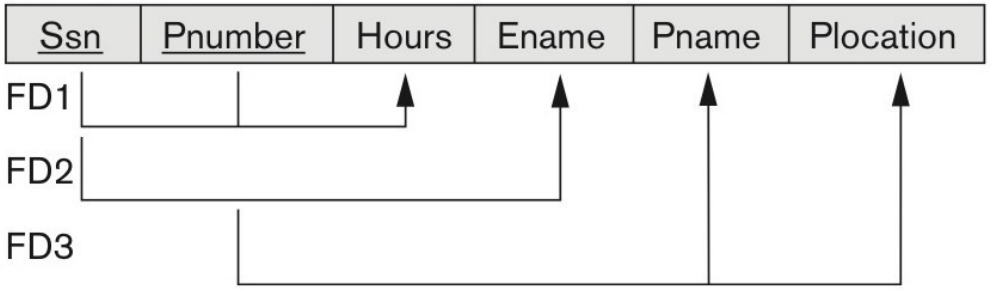
(a)

EMP_DEPT



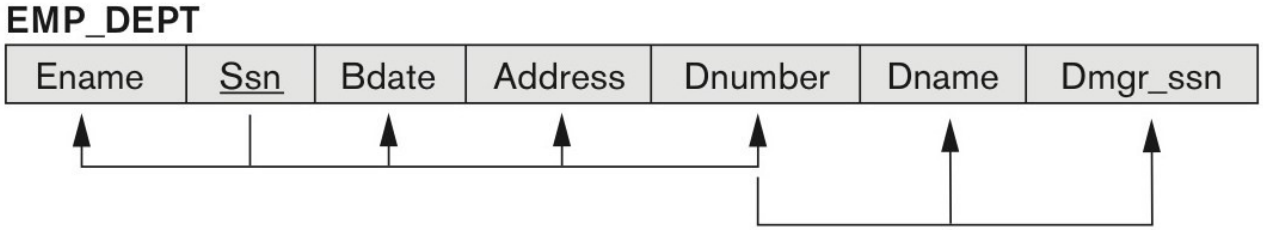
(b)

EMP_PROJ

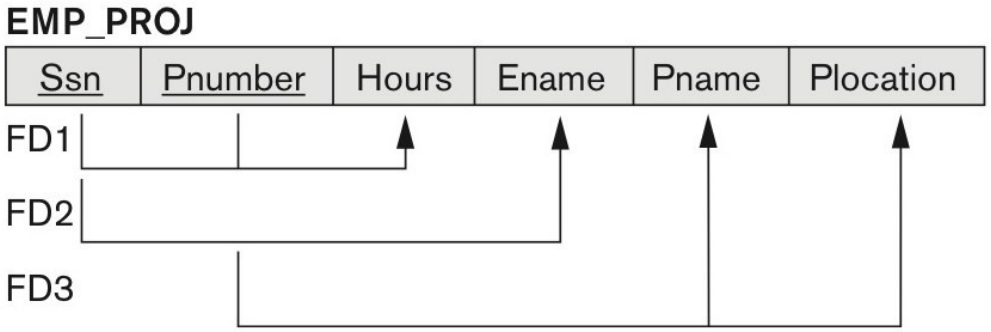


Any concerns here?

(a)



(b)



Any concerns here?

EMP_DEPT: mixing attributes of employees & departments

EMP_PROJ: mixes attributes of employees, projects & works_on

Figure 14.4
Sample states for EMP_DEPT and EMP_PROJ resulting from applying NATURAL JOIN to the relations in Figure 14.2. These may be stored as base relations for performance reasons.

Redundancy

EMP_DEPT

Ename	Ssn	Bdate	Address	Dnumber	Dname	Dmgr_ssn
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5	Research	333445555
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5	Research	333445555
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4	Administration	987654321
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX	4	Administration	987654321
Narayan, Ramesh K.	666884444	1962-09-15	975 FireOak, Humble, TX	5	Research	333445555
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX	5	Research	333445555
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX	4	Administration	987654321
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX	1	Headquarters	888665555

Redundancy Redundancy

EMP_PROJ

Ssn	Pnumber	Hours	Ename	Pname	Plocation
123456789	1	32.5	Smith, John B.	ProductX	Bellaire
123456789	2	7.5	Smith, John B.	ProductY	Sugarland
666884444	3	40.0	Narayan, Ramesh K.	ProductZ	Houston
453453453	1	20.0	English, Joyce A.	ProductX	Bellaire
453453453	2	20.0	English, Joyce A.	ProductY	Sugarland
333445555	2	10.0	Wong, Franklin T.	ProductY	Sugarland
333445555	3	10.0	Wong, Franklin T.	ProductZ	Houston
333445555	10	10.0	Wong, Franklin T.	Computerization	Stafford
333445555	20	10.0	Wong, Franklin T.	Reorganization	Houston
999887777	30	30.0	Zelaya, Alicia J.	Newbenefits	Stafford
999887777	10	10.0	Zelaya, Alicia J.	Computerization	Stafford
987987987	10	35.0	Jabbar, Ahmad V.	Computerization	Stafford
987987987	30	5.0	Jabbar, Ahmad V.	Newbenefits	Stafford
987654321	30	20.0	Wallace, Jennifer S.	Newbenefits	Stafford
987654321	20	15.0	Wallace, Jennifer S.	Reorganization	Houston
888665555	20	Null	Borg, James E.	Reorganization	Houston

1.2 Redundant Information in Tuples and Update Anomalies

- Information is stored redundantly
  - Wastes storage
  - Causes problems with update anomalies
    - Insertion anomalies
    - Deletion anomalies
    - Modification anomalies

EXAMPLE OF AN INSERT ANOMALY

Consider the relation:

EMP_PROJ(Emp#, Proj#, Ename, Pname, No_hours)

Insert Anomaly:

Cannot insert a project unless an employee is assigned to it

Conversely

Cannot insert an employee unless an he/she is assigned to a project

EXAMPLE OF A DELETE ANOMALY

Consider the relation:

EMP_PROJ(Emp#, Proj#, Ename, Pname, No_hours)

Delete Anomaly:

When a project is deleted, it will result in deleting all the employees who work on that project.

Alternately, if an employee is the sole employee on a project, deleting that employee would result in deleting the corresponding project.

EXAMPLE OF AN UPDATE ANOMALY

Consider the relation:

EMP_PROJ(Emp#, Proj#, Ename, Pname, No_hours)

Update Anomaly:

Changing the name of project number P1 from “Billing” to “Customer-Accounting” may cause this update to be made for all 100 employees working on project P1.

Guideline for Redundant Information in Tuples and Update Anomalies

GUIDELINE 2:

- Design a schema that does not suffer from the insertion, deletion and update anomalies
- If there are any anomalies present, then note them so that applications can be made to take them into account

1.3 Null Values in Tuples

GUIDELINE 3:

- Relations should be designed such that their tuples will have as few NULL values as possible
- Attributes that are NULL frequently could be placed in separate relations (with the primary key)

Reasons for nulls; different meanings for null:

- Attribute not applicable or invalid [visa status to US students]
- Attribute value unknown [DOB of an employee]
- Value is known but absent; it has not been recorded yet [phone # of employee]

1.4 Generation of Spurious Tuples – avoid at any cost

Bad designs for a relational database may result in erroneous results for certain JOIN operations

GUIDELINE 4:

> No spurious tuples should be generated by doing a natural-join of any relations.

(a)

EMP_LOCS

Ename	Plocation

P.K.

EMP_PROJ1

Ssn	Pnumber	Hours	Pname	Plocation

P.K.

(b)

EMP_LOCS

Ename	Plocation
Smith, John B.	Bellaire
Smith, John B.	Sugarland
Narayan, Ramesh K.	Houston
English, Joyce A.	Bellaire
English, Joyce A.	Sugarland
Wong, Franklin T.	Sugarland
Wong, Franklin T.	Houston
Wong, Franklin T.	Stafford
Zelaya, Alicia J.	Stafford
Jabbar, Ahmad V.	Stafford
Wallace, Jennifer S.	Stafford
Wallace, Jennifer S.	Houston
Borg, James E.	Houston

EMP_PROJ1

Ssn	Pnumber	Hours	Pname	Plocation
123456789	1	32.5	ProductX	Bellaire
123456789	2	7.5	ProductY	Sugarland
666884444	3	40.0	ProductZ	Houston
453453453	1	20.0	ProductX	Bellaire
453453453	2	20.0	ProductY	Sugarland
333445555	2	10.0	ProductY	Sugarland
333445555	3	10.0	ProductZ	Houston
333445555	10	10.0	Computerization	Stafford
333445555	20	10.0	Reorganization	Houston
999887777	30	30.0	Newbenefits	Stafford
999887777	10	10.0	Computerization	Stafford
987987987	10	35.0	Computerization	Stafford
987987987	30	5.0	Newbenefits	Stafford
987654321	30	20.0	Newbenefits	Stafford
987654321	20	15.0	Reorganization	Houston
888665555	20	NULL	Reorganization	Houston

	Ssn	Pnumber	Hours	Pname	Plocation	Ename
	123456789	1	32.5	ProductX	Bellaire	Smith, John B.
*	123456789	1	32.5	ProductX	Bellaire	English, Joyce A.
	123456789	2	7.5	ProductY	Sugarland	Smith, John B.
*	123456789	2	7.5	ProductY	Sugarland	English, Joyce A.
*	123456789	2	7.5	ProductY	Sugarland	Wong, Franklin T.
	666884444	3	40.0	ProductZ	Houston	Narayan, Ramesh K.
*	666884444	3	40.0	ProductZ	Houston	Wong, Franklin T.
*	453453453	1	20.0	ProductX	Bellaire	Smith, John B.
	453453453	1	20.0	ProductX	Bellaire	English, Joyce A.
*	453453453	2	20.0	ProductY	Sugarland	Smith, John B.
	453453453	2	20.0	ProductY	Sugarland	English, Joyce A.
*	453453453	2	20.0	ProductY	Sugarland	Wong, Franklin T.
*	333445555	2	10.0	ProductY	Sugarland	Smith, John B.
*	333445555	2	10.0	ProductY	Sugarland	English, Joyce A.
	333445555	2	10.0	ProductY	Sugarland	Wong, Franklin T.
*	333445555	3	10.0	ProductZ	Houston	Narayan, Ramesh K.
	333445555	3	10.0	ProductZ	Houston	Wong, Franklin T.
	333445555	10	10.0	Computerization	Stafford	Wong, Franklin T.
*	333445555	20	10.0	Reorganization	Houston	Narayan, Ramesh K.
	333445555	20	10.0	Reorganization	Houston	Wong, Franklin T.

*
*
*

Additional tuples that were not there in Emp_proj is here, they are called spurious tuples

2. Functional Dependencies

Functional dependencies (FDs)

- Are used to specify *formal measures* of the "goodness" of relational designs
- And keys are used to define **normal forms** for relations
- Are **constraints** that are derived from the *meaning* and *interrelationships* of the data attributes

A set of attributes X *functionally determines* a set of attributes Y if the value of X determines a unique value for Y

2.1 Defining Functional Dependencies

X → Y holds if whenever two tuples have the same value for X, they *must have* the same value for Y

> For any two tuples t1 and t2 in any relation instance r(R): If t1[X]=t2[X], *then* t1[Y]=t2[Y]

X → Y in R specifies a *constraint* on all relation instances r(R)

Written as X → Y; can be displayed graphically on a relation schema as in Figures; denoted by the arrow →

FDs are derived from the real-world constraints on the attributes

Examples of FD constraints (1)

Social security number determines employee name

SSN → ENAME

Project number determines project name and location

PNUMBER → {PNAME, PLOCATION}

Employee ssn and project number determines the hours per week that the employee works on the project

{SSN, PNUMBER} → HOURS

Examples of FD constraints (1)

Social security number determines employee name

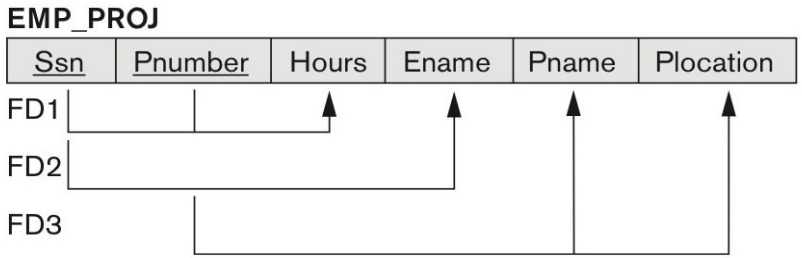
SSN → ENAME

Project number determines project name and location

PNUMBER → {PNAME, PLOCATION}

Employee ssn and project number determines the hours per week that the employee works on the project

{SSN, PNUMBER} → HOURS



Examples of FD constraints (2)

An FD is a property of the attributes in the schema R

The constraint must hold on *every* relation instance r(R)

If K is a key of R, then K functionally determines all attributes in R

Defining FDs from instances

Note that in order to define the FDs, we need to understand the meaning of the attributes involved and the relationship between them.

Given the instance (population) of a relation, all we can conclude is that an FD *may exist* between certain attributes.

What we can definitely conclude is – that certain FDs *do not exist* because there are tuples that show a violation of those dependencies.

Ruling Out FDs

Note that given the state of the TEACH relation, we can say that the FD: Text → Course may exist. However, the FDs Teacher → Course, Teacher → Text and Course → Text are ruled out.

TEACH

Teacher	Course	Text
Smith	Data Structures	Bartram
Smith	Data Management	Martin
Hall	Compilers	Hoffman
Brown	Data Structures	Horowitz

What FDs may exist?

A relation *R*(A, B, C, D) with its extension.
Which FDs *may exist* in this relation?

A	B	C	D
a1	b1	c1	d1
a1	b2	c2	d2
a2	b2	c2	d3
a3	b3	c4	d3

What FDs may exist?

A relation *R*(A, B, C, D) with its extension.
Which FDs *may exist* in this relation?

A	B	C	D
a1	b1	c1	d1
a1	b2	c2	d2
a2	b2	c2	d3
a3	b3	c4	d3

B → C; C → B; {A,B} → C; {A,B} → D; {C,D}→ B

How about A → B? B→ A? D → C?

Bibliography / Acknowledgements

Instructor materials from Elmasri & Navathe 7e


pk.profgiri


Ponnurangam.kumaraguru


/in/ponguru


ponguru

Thank you for attending the class!!!


pk.guru@iiit.ac.in