CS4.301 Data & Applications

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

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

Definitions of Keys and Attributes Participating in Keys (2)

A **Prime attribute** must be a member of *some* candidate key

A **Nonprime attribute** is not a prime attribute—that is, it is not a member of any candidate key.

Prime attributes

Consider the relation:

`R(A, B, C, D)`

and suppose the **candidate keys** are:

- `{A, B}`
- `{C, D}`

Then:

- **Prime attributes:** A, B, C, D (since all are part of some candidate key).
- **Non-prime attributes:** None (since every attribute participates in at least one key).

Prime attributes

Consider the relation:

`R(A, B, C, D)`

and suppose the **candidate keys** are:

- `{A, B}`
- `{C, D}`

Then:

- **Prime attributes:** A, B, C, D (since all are part of some candidate ke
- **Non-prime attributes:** None (since every attribute participates in a

Another example:

`R(A, B, C, D)` with **candidate key** `{A, B}` only.

Then:

- **Prime attributes:** A, B
- **Non-prime attributes:** C, D

3.4 First Normal Form

Disallows

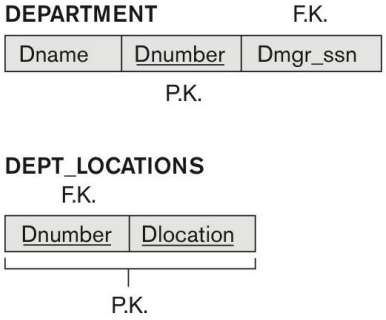
- composite attributes
- multivalued attributes
- **nested relations**; attributes whose values for an *individual tuple* are non-atomic

Considered to be part of the definition of a relation

Most RDBMSs allow only those relations to be defined that are in First Normal Form

1NF

2 = Redundancy, Dnumber & Dlocation primary key



DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocation
Research	5	333445555	Bellaire
Research	5	333445555	Sugarland
Research	5	333445555	Houston
Administration	4	987654321	Stafford
Headquarters	1	888665555	Houston

1 = Two 1NF relations

3 = If the maximum number of values (n) for location is known, replace it with n attributes
e.g. Only 3 locations for the company – Dlocation1, Dlocation2, Dlocation3
Introducing NULL if most departments have fewer than 3 locations
Hard to query, e.g. List the departments that have 'Bellaire' as one of the locations
1st option is commonly used one

Fully functional dependency

If X and Y are an attribute set of a relation, Y is fully functional dependent on X, if Y is functionally dependent on X but not on any proper subset of X.

e.g. In the relation ABC->D, attribute D is fully functionally dependent on ABC and not on any proper subset of ABC. That means that subsets of ABC like AB, BC, A, B, etc cannot determine D.

supplier_id	item_id	price
1	1	540
2	1	545
1	2	200
2	2	201
1	1	540
2	2	201
3	1	542

{ supplier_id , item_id } -> price

https://www.geeksforgeeks.org/differentiate-between-partial-dependency-and-fully-functional-dependency/

Second Normal Form (2)

A relation schema R is in **second normal form (2NF)** if every non-prime attribute A in R is fully functionally dependent on the primary key

A **Prime attribute** must be a member of *some* candidate key

A **Nonprime attribute** is not a prime attribute—that is, it is not a member of any candidate key.

R can be decomposed into 2NF relations via the process of 2NF normalization or “second normalization”

Normalizing into 2NF

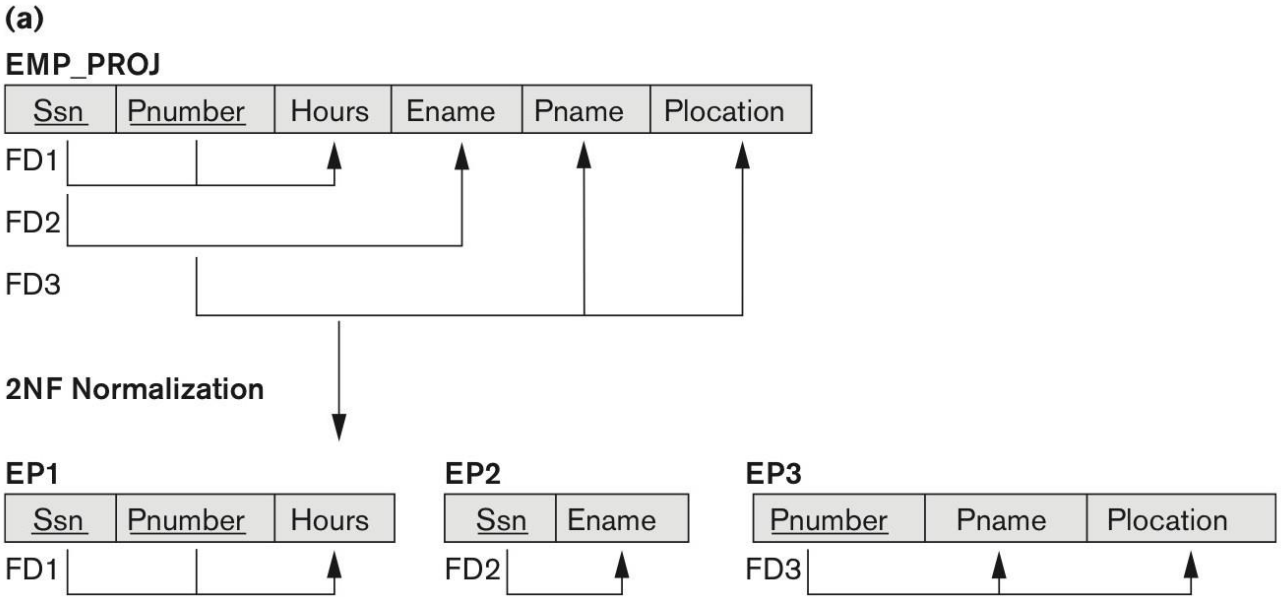


Figure 14.11
Normalizing into 2NF and 3NF. (a) Normalizing EMP_PROJ into 2NF relations.

EP1, EP2, EP3 are fully functionally dependent

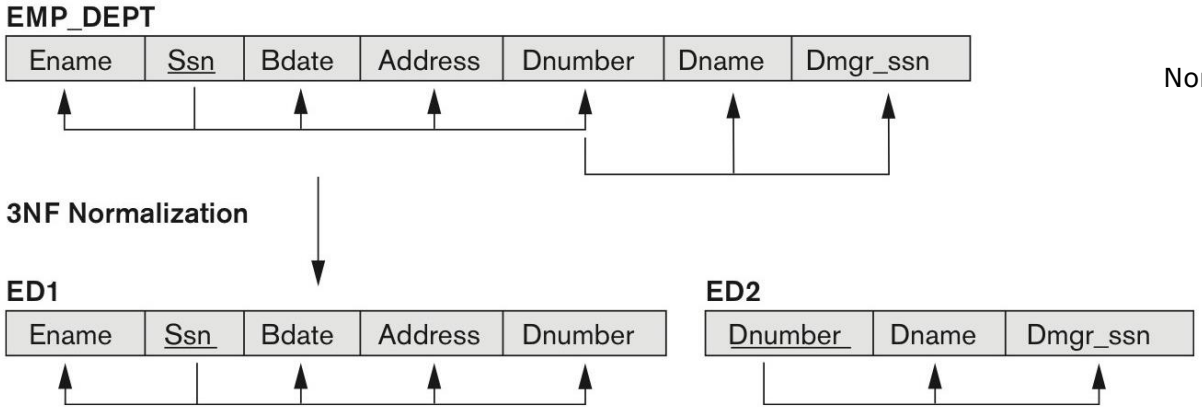
Third normal form

Transitive Dependency

X → Y in R is transitive dependency, if there exists a set of attributes Z in R that is neither a candidate key nor a subset of any key of R and both X → Z & Z → Y hold.

Third normal form

R is in 3NF if it satisfies 2NF and no nonprime attribute of R is transitively dependent on the primary key



Normalizing EMP_DEPT into 3NF relations.

ED1 & ED2 represent independent facts about employees & departments

5. BCNF (Boyce-Codd Normal Form)

A relation schema R is in **Boyce-Codd Normal Form (BCNF)** if whenever an **FD X → A** holds in R, then **X is a superkey** of R

Refresher: Roll number: Candidate key; Roll number + Name: Super key

Each normal form is strictly stronger than the previous one

- Every 2NF relation is in 1NF
- Every 3NF relation is in 2NF
- Every BCNF relation is in 3NF

There exist relations that are in 3NF but not in BCNF

Hence BCNF is considered a stronger form of 3NF

The goal is to have each relation in BCNF (or 3NF)

Interested in working with Precog?

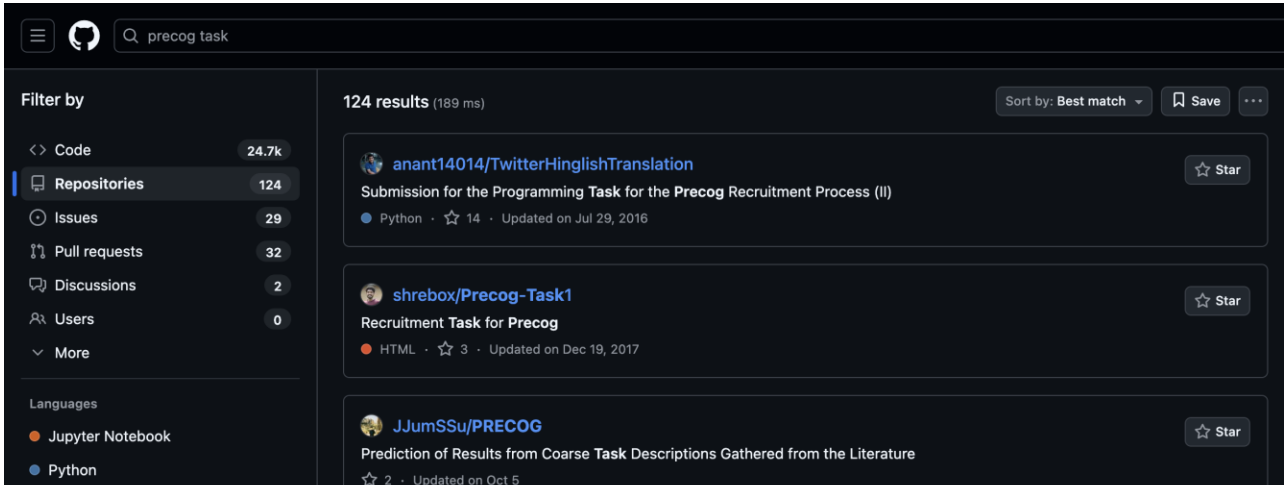
Look for an email in Jan 1st week after start of classes

Areas / Questions: NLP & Responsible AI; Computer Vision; Graphs; Math / Quant

Process: Apply (SOP, CV, etc.) – Task – Technical Interview

What to prepare? Pick an area from above & try out topics

All process done by mid sem



Activity: 13th Nov

(0) Write 2 prime & 2 full dependency attributes in Infinium DB
(1) Write 2 non-prime & 2 partial dependency attributes in Infinium DB

Relation: EMP_PROJECT

| EmpNo | EmpName | ProjNo | ProjName | DeptNo | DeptName | HoursWorked | DeptLocation |

Given information:

1. Each employee works in one department, but a department can have many employees.
2. Each project is controlled by one department.
3. An employee can work on multiple projects, and a project can have multiple employees.
4. `HoursWorked` is the number of hours that an employee works on a particular project.

(2a) Identify all functional dependencies (FDs).

(2b) Find the candidate keys.

(2c) Convert the relation into 1NF, 2NF, 3NF, and BCNF.

(2d) Draw the resulting schemas at each stage.

This Lecture

Administrativia

How was the Quiz #3?

Make up quiz on the day of end-sem exams; if you take the make up quiz it will be counted one of the 3

Same day as end sem at 8 – 9PM

Superkey

Super Key is an attribute (or set of attributes) that is used to uniquely identify all attributes in a relation.

Is a set of attributes SK of R with the following condition:

- No two tuples in any valid relation state r(R) will have the same value for SK
- That is, for any distinct tuples t1 and t2 in r(R), t1[SK] $\neq$ t2[SK]
- This condition must hold in *any valid state* r(R)

Think of superkey = any combination of attributes that guarantees uniqueness, even if it has extra unnecessary attributes.

Any set of attributes that can uniquely identify a row.

Running example

STUDENT(RollNo, Email, Phone, Name, Dept)

Assumptions:

RollNo is unique for every student

Email is unique

Phone is unique

Name and Dept are *not* unique

Superkeys

{RollNo}

{Email}

{Phone}

{RollNo, Name}

{Email, Dept}

{RollNo, Email, Phone, Name}

If it uniquely identifies a row, it is a superkey, even if it is unnecessarily large.

A superkey is like using your passport + Aadhaar + driving license to prove your identity, overkill but still unique.

Candidate Key

Minimal superkeys

Candidate keys = Superkeys with nothing extra. If you remove any attribute, it stops being unique.

STUDENT(RollNo, Email, Phone, Name, Dept) ??????

Candidate Key

Minimal superkeys

Candidate keys = Superkeys with nothing extra. If you remove any attribute, it stops being unique.

STUDENT(RollNo, Email, Phone, Name, Dept) ??????

Candidate Keys: {RollNo}, {Email}, {Phone}

Not Candidate Keys: {RollNo, Name}, {Email, Dept}

Candidate keys are like the smallest set of documents you need to prove your identity

Primary Key

Chosen candidate key

You pick ONE candidate key to be the official identifier of the table

General rule: Choose as primary key the smallest of the candidate keys (in terms of size)

We choose RollNo as the Primary Key

Email and Phone remain candidate keys

Only one becomes the primary key

Primary key is the official ID card you choose from all the valid options (candidate keys)

Concept	Meaning	Example(s) from STUDENT
Superkey	Any set of attributes that uniquely identifies a row	{RollNo}, {Email}, {RollNo, Name}, {Email, Dept}, ...
Candidate Key	Minimal superkey	{RollNo}, {Email}, {Phone}
Primary Key	Chosen candidate key	{RollNo}

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.

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.

E.Fname	E.Lname	S.Fname	S.Lname
John	Smith	Franklin	Wong
Franklin	Wong	James	Borg
Alicia	Zelaya	Jennifer	Wallace
Jennifer	Wallace	James	Borg
Ramesh	Narayan	Franklin	Wong
Joyce	English	Franklin	Wong
Ahmad	Jabbar	Jennifer	Wallace

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');
```

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');
```

Nested Queries (cont'd.)

Use tuples of values in comparisons

Place them within parentheses

```
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)
```

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

```
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)
```

Find employees who share at least one (project, hours) pair with employee 123456789.

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

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

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 ???? Ideas?

```
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);
```

Correlated nested query

Evaluated once for each tuple in the outer query

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);

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)
```

Aggregate Functions in SQL

Used to summarize information from multiple tuples into a single-tuple summary

Built-in aggregate functions

- **COUNT**, **SUM**, **MAX**, **MIN**, and **AVG**

Grouping

- Create subgroups of tuples before summarizing

To select entire groups, `HAVING` clause is used

Aggregate functions can be used in the `SELECT` clause or in a `HAVING` clause

Renaming Results of Aggregation

```
SELECT SUM(Salary),
MAX(Salary),
MIN(Salary), AVG(Salary)
FROM EMPLOYEE;
```

Renaming Results of Aggregation

```
SELECT SUM(Salary),
MAX(Salary),
MIN(Salary), AVG(Salary)
FROM EMPLOYEE;
```

```
mysql> SELECT SUM(Salary), MAX(Salary), MIN(Salary), AVG(Sal
ary) FROM EMPLOYEE;
+-------------+-------------+-------------+-------------+
| SUM(Salary) | MAX(Salary) | MIN(Salary) | AVG(Salary) |
+-------------+-------------+-------------+-------------+
|      281000 |       55000 |       25000 |  35125.0000 |
+-------------+-------------+-------------+-------------+
1 row 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)
```

Comparisons Involving NULL

SQL allows queries that check whether an attribute value is `NULL`

`IS` or `IS NOT NULL`

Query 18. Retrieve the names of all employees who do not have supervisors.

Comparisons Involving NULL

SQL allows queries that check whether an attribute value is `NULL`

`IS` or `IS NOT NULL`

Query 18. Retrieve the names of all employees who do not have supervisors.

```
Q18:  SELECT  Fname, Lname
      FROM    EMPLOYEE
      WHERE   Super_ssn IS NULL;
```

```
mysql> Select fname, lname from employee where super_ssn IS null;
+-------+-------+
| fname | lname |
+-------+-------+
| James | Borg  |
+-------+-------+
1 row in set (0.00 sec)
```

IS & IS NOT

Retrieve the names of all employees who have supervisors

IS & IS NOT

Select fname, lname from employee where super_ssn IS NOT null;

```
mysql> Select fname, lname from employee where super_ssn IS NOT null;
+----------+---------+
| fname    | lname   |
+----------+---------+
| John     | Smith   |
| Franklin | Wong    |
| Joyce    | English |
| Ramesh   | Narayan |
| Jennifer | Wallace |
| Ahmad    | Jabbar  |
| Alicia   | Zelaya  |
+----------+---------+
7 rows in set (0.00 sec)

mysql>
```

Grouping: The GROUP BY Clause

Partition relation into subsets of tuples

- Based on **grouping attribute(s)**
- Apply function to each such group independently

`GROUP BY` clause

- Specifies grouping attributes

Group BY example

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

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.

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)
```

Specification of Views in SQL (cont'd.)

Once a View is defined, SQL queries can use the View relation in the FROM clause

View is always up-to-date

- Responsibility of the DBMS and not the user

`DROP VIEW` command

- Dispose of a view

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

Activity 20th Nov

1. Try Nested Query in Slide 31
2. Try Nested Query in Slide 33
3. Retrieve the name of each employee who has a dependent with the same first name and is the same sex as the employee.
4. Find the sum of the salaries of all employees of the ‘Research’ department, as well as the maximum salary, the minimum salary, and the average salary in this department.

Nested Queries (cont'd.)

Avoid potential errors and ambiguities

Create tuple variables (aliases) for all tables referenced in SQL query

Query 16. Retrieve the name of each employee who has a dependent with the same first name and is the same sex as the employee.

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);

```
mysql> Select e.fname, e.lname from employee as e w
here e.ssn in (select essn from dependent as d wher
e e.fname=d.dependent_name and e.sex=d.sex);
Empty set (0.00 sec)
```

Aggregate Functions in SQL (cont'd.)

Query 20. Find the sum of the salaries of all employees of the 'Research' department, as well as the maximum salary, the minimum salary, and the average salary in this department.

```
SELECT SUM(Salary),
MAX(Salary),
MIN(Salary), AVG(Salary)
FROM (EMPLOYEE join
department on
dno=dnumber) where
dname='research';
```

```
mysql> SELECT SUM(Salary), MAX(Salary), MIN(Salary), AVG(Sal
ary) FROM (EMPLOYEE join department on dno=dnumber) where dn
ame='research';
+-------------+-------------+-------------+-------------+
| SUM(Salary) | MAX(Salary) | MIN(Salary) | AVG(Salary) |
+-------------+-------------+-------------+-------------+
|      133000 |       40000 |       25000 |  33250.0000 |
+-------------+-------------+-------------+-------------+
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)
```

Try it yourself!

Query 12. Retrieve all employees whose address is in Houston, Texas.

Query 12A. Find all employees who were born during the 1950s.

Query 14. Retrieve all employees in department 5 whose salary is between $30,000 and $40,000.

Query 12. Retrieve all employees whose address is in Houston, Texas.

```
Q12:    SELECT    Fname, Lname
        FROM      EMPLOYEE
        WHERE     Address LIKE '%Houston,TX%';
```

Query 12A. Find all employees who were born during the 1950s.

```
Q12:    SELECT    Fname, Lname
        FROM      EMPLOYEE
        WHERE     Bdate LIKE '_ _ 7 _ _ _ _ _ _ _';
```

Query 14. Retrieve all employees in department 5 whose salary is between $30,000 and $40,000.

```
Q14:    SELECT    *
        FROM      EMPLOYEE
        WHERE     (Salary BETWEEN 30000 AND 40000) AND Dno = 5;
```

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