# SQL:
# A COMMERCIAL DATABASE LANGUAGE

## Complex Queries, Aggregate Functions and Grouping

# Outline of Chapters 8, 9

1. Introduction
2. Data Definition, Basic Constraints, and Schema Changes
3. Basic Queries
4. More complex Queries
5. Aggregate Functions and Grouping
6. Summary of SQL queries
7. Data Change statements
8. Views
9. Complex Constraints
10. Database Programming

# 4. More complex queries

- Because of the generality and expressive power of SQL , there are many additional features (e.g. query nesting, and grouping/aggregation) that allow users to specify more complex queries.

# 4.1 Nested queries (1)

- A complete SELECT query, called a *nested query*, can be specified within the WHERE-clause of another query, called the *outer query*.

- Example:

Query 1: Retrieve the name and address of all employees who work for the 'Research' department.

```
SELECT    FNAME, LNAME, ADDRESS
FROM      EMPLOYEE
WHERE     DNO    IN    (SELECT      DNUMBER
                        FROM        DEPARTMENT
                        WHERE       DNAME='Research');
```

- The nested query selects the number of the ‘Research' department.
- The outer query selects an EMPLOYEE tuple if its DNO value is in the result of the nested query.
- The comparison operator IN compares a value v with a set (or multi-set) of values V, and evaluates to TRUE if v is one of the elements in V.

# 4.1 Nested queries (2)

- Example:

  Query 4: Make a list of all project numbers for projects that involve an employee whose last name is 'Smith' as a worker or as a manager of the department that controls the project.

```
(SELECT   DISTINCT PNUMBER
FROM      PROJECT
WHERE     PNUMBER IN
          (SELECT   PNUMBER
          FROM      PROJECT, DEPARTMENT, EMPLOYEE
          WHERE     DNUM=DNUMBER AND MGRSSN=SSN
                    AND   LNAME='Smith')
          OR
               PNUMBER IN
          (SELECT   PNO
          FROM      WORKS_ON, EMPLOYEE
          WHERE     ESSN=SSN AND LNAME='Smith');
```

# 4.1 Nested queries (3)

- The IN comparison operator can also compare a tuple of values in parentheses with a set of *union-compatible* tuples.

- Example:

```
SELECT      DISTINCT ESSN
FROM        WORKS_ON
WHERE       (PNO, HOURS) IN
                (SELECT  PNO, HOURS
                 FROM    WORKS_ON
                 WHERE   ESSN = ‘123456789');
```

Select the social security number of all employees who work the same (project, hours) combination on a project that the employee whose SSN is ‘123456789’ works on.

# 4.1 Nested queries (4)

- Other comparison operators can also be used to *compare a single value v (typically an attribute name) to a set or multiset V (typically a nested query).*
- **θ ANY** ( **θ SOME** ) and
  **θ ALL**
  where θ is one of **=, <, <=, >, >=, <>**.

  **= ANY** (or **= SOME**) is equivalent to **IN**.
- Example**:** Retrieve the names of the employees whose salary is greater than the salary of all the employees in department number 5.

```
SELECT   LNAME, FNAME
FROM     EMPLOYEE
WHERE    SALARY > ALL  (SELECT  SALARY
                        FROM    EMPLOYEE
                        WHERE   DNO = 5);
```

# 4.1 Nested queries (5)

- In general, we can have *several levels of nested queries*.

- Ambiguity can arise among attribute names if the same attribute exists in relations in the FROM clauses of both the outer query and a nested query.

Rule:

- A reference to an *unqualified attribute* refers to the relation declared in the *innermost nested query.*

- To refer to an attribute of a relation specified in an outer query, we can *specify and refer to an alias for that relation*.

-- examples follow.

# 4.1 Nested queries (6)

- Example:

```
SELECT   LNAME, FNAME
FROM     EMPLOYEE
WHERE    SALARY > ALL
         (SELECT    SALARY
         FROM       EMPLOYEE
         WHERE      DNO = 5);
```

Attributes SALARY and DNO in the nested query refer to the EMPLOYEE relation declared in the nested query.

# 4.2 Correlated nested queries (1)

- If a condition in the WHERE-clause of a *nested query* references an attribute of a relation declared in the *outer query*, the two queries are said to be *correlated*.
- The result of a correlated nested query is *different for each tuple (or combination of tuples) of the relation(s) of the outer query*.
- We can understand a correlated query, by considering that *the nested query is evaluated once for each tuple (or combination of tuples) of the outer query*.

# 4.2 Correlated nested queries (2)

- Example:

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

```
SELECT   E.FNAME, E.LNAME
FROM     EMPLOYEE AS E
WHERE    E.SSN IN
            (SELECT  ESSN
             FROM    DEPENDENT
             WHERE   SEX = E.SEX AND
                     E.FNAME = DEPENDENT_NAME);
```

We can think that Query 16 is evaluated as follows: for each EMPLOYEE tuple (outer query) evaluate the nested query, which retrieves the ESSN values for all DEPENDENT tuples with the same sex and first name as the EMPLOYEE tuple.

# 4.2 Correlated nested queries (3)

- In general, a query with nested SELECT-FROM-WHERE blocks that uses the IN comparison operator can be expressed using a single query block.

- Example:
  Query 16 can be rewritten as follows:

```
SELECT    E.FNAME, E.LNAME
FROM      EMPLOYEE AS E, DEPENDENT AS D
WHERE     D.ESSN = E.SSN AND D.SEX = E.SEX AND
          E.FNAME = D.DEPENDENT_NAME);
```

# 4.3 The EXISTS function (1)

- The EXISTS function in SQL is used to check whether the result of a correlated nested query is empty (contains no tuples) or not.
- Example: Query 16 can be rewritten as follows:

```
SELECT   E.FNAME, E.LNAME
FROM     EMPLOYEE AS E
WHERE    EXISTS
         (SELECT   *
          FROM     DEPENDENT
          WHERE    E.SSN = ESSN AND E.SEX = SEX AND
                   E.FNAME = DEPENDENT_NAME);
```

- We can think that this query is evaluated as follows: *for each EMPLOYEE tuple (outer query)* the nested query which retrieves all DEPENDENT tuples with the same social security number, sex, and first name as the EMPLOYEE tuple is evaluated. If the *result is not empty*, select the EMPLOYEE tuple.

# 4.3 The EXISTS function (2)

- The function NOT EXISTS can also be used.

  EXISTS(Q) returns TRUE if *there is at least one tuple in the result of Q*, and FALSE otherwise.

  *NOT EXISTS(Q)* returns TRUE if *there are no tuples in the result of Q*, and FALSE otherwise.

- Example**:** Query 6. Retrieve the names of employees who have no dependents.

```
SELECT   FNAME, LNAME
FROM     EMPLOYEE
WHERE    NOT EXISTS  (SELECT    *
                      FROM      DEPENDENT
                      WHERE     SSN=ESSN) ;
```

# 4.3 The EXISTS function (3)

- Another example**:** Query 7. List the names of managers who have at least one dependent.

```
SELECT   FNAME, LNAME
FROM     EMPLOYEE
WHERE    EXISTS  (SELECT   *
                  FROM     DEPENDENT
                  WHERE    SSN = ESSN)
         AND
         EXISTS  (SELECT   *
                  FROM     DEPARTMENT
                  WHERE    SSN = MGRSSN);
```

Here we specify two nested correlated queries. Can we specify the query using one nested query? No nested queries?

# 4.3 The EXISTS function (*1)

- Another example:

Query 7. List the names of managers who have at least one dependent.

```
SELECT     FNAME, LNAME
FROM       EMPLOYEE
WHERE      EXISTS
           (SELECT  *
           FROM     DEPARTMENT, DEPENDENT
           WHERE    SSN = ESSN AND SSN = MGRSSN );
```

# 4.3 The EXISTS function (*2)

- Another example:

Query 7. List the names of managers who have at least one dependent.

```
SELECT     FNAME, LNAME
FROM       EMPLOYEE, DEPARTMENT, DEPENDENT
WHERE      SSN = ESSN AND SSN = MGRSSN;
```

Note that, in contrast to the previous rewritings of Query 7, this one may contain duplicates in the answer even if there are no two employees with the same FNAME and LNAME (Why?)

# 4.3 The EXISTS function (4)

- Example (cont.):

  Query 3. Retrieve the name of each employee who works on all the projects.

  We can express this query by using one level of nesting and the set theoretic operation of set difference.

```
SELECT   FNAME, LNAME
FROM     EMPLOYEE
WHERE    NOT EXISTS
   (       (SELECT    PNUMBER
            FROM      PROJECT)
            EXCEPT
           (SELECT    PNO
            FROM      WORKS_ON
            WHERE     ESSN = SSN)  );
```

# 4.3 The EXISTS function (5)

- Example (cont.):

  Query 3. Retrieve the name of each employee who works on all the projects.

  Another option for Query 3 is to use a two-level nesting.

```
SELECT  FNAME, LNAME
FROM    EMPLOYEE
WHERE   NOT EXISTS
            (SELECT   *
             FROM     PROJECT
             WHERE    NOT EXISTS
               (SELECT *
                FROM     WORKS_ON
                WHERE    ESSN = SSN AND PNO = PNUMBER));
```

# 5. Aggregate functions and grouping

- Because grouping and aggregation are required in many database applications, SQL has features that incorporate these concepts.

- SQL has a number of built-in aggregate functions:
  **COUNT**, **SUM**, **MAX**, **MIN**, and **AVG** .

  The COUNT function returns the number of tuples or values as specified in the query.

  SUM, MAX, MIN, and AVG are applied to a multiset of numeric values and return, respectively, the sum, maximum value, minimum value, and average of those values.

  MIN and MAX can be used with non numeric attributes if their domains have a total ordering (e.g. domain DATE).

# 5.1 Aggregate queries (1)

- Example:

  Query 19. Find the sum of the salaries, the maximum salary, the minimum salary, and the average salary among all employees.

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

Some SQL implementations *may not allow more than one function* in the SELECT-clause

# 5.1 Aggregate queries (2)

- Example:

  Query 20. Find the sum of the salaries, the maximum salary, the minimum salary, and the average salary among employees who work for the 'Research' department .

```
SELECT    SUM(SALARY), MAX(SALARY), MIN(SALARY),
          AVG(SALARY)
FROM      EMPLOYEE, DEPARTMENT
WHERE     DNO = DNUMBER AND DNAME='Research' ;
```

# 5.1 Aggregate queries (3)

- Example:

  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’;
```

**COUNT(*)** returns the number of rows (tuples) in the result of the query.

# 5.1 Aggregate queries (4)

- The COUNT function can also be used to count values in a column rather than tuples.

- Example**:**
  Query 23. Count the number of distinct salary values in the database.

```
SELECT   COUNT ( DISTINCT SALARY)
FROM     EMPLOYEE;
```

If **COUNT**(SALARY) had been used instead of
**COUNT ( DISTINCT** SALARY**)**,
all the non-null SALARY values in the EMPLOYEE relation
(including duplicates) would have been counted.

# 5.1 Aggregate queries (5)

- Aggregate queries are one-tuple queries. They can be nested in the WHERE clause of an outer query to retrieve a summary value from the database.
- Example:

  Query 5. Retrieve the names of all employees who have two or more dependents.

```
SELECT   LNAME, FNAME
FROM     EMPLOYEE
WHERE    (SELECT COUNT(*)
         FROM DEPENDENT
         WHERE SSN = ESSN) >= 2;
```

# 5.2 Grouping queries (1)

- In many cases, we want to apply the aggregate functions *to subgroups of tuples in a relation.*
- Each subgroup of tuples consists of the set of tuples that have *the same value* for some attribute(s), called *grouping attribute(s)*.
- The aggregate function is applied *to each subgroup independently*.
- SQL has a **GROUP BY**-clause for specifying the grouping attributes.
- The grouping attributes should *also appear in the SELECT-clause*, *so* that the value resulting from applying each aggregate function to a group of tuples appears along with values of the grouping attribute(s).

# 5.2 Grouping queries (2)

- Example**:**

  Query 24. For each department, retrieve the department number, the number of employees in the department, and their average salary.

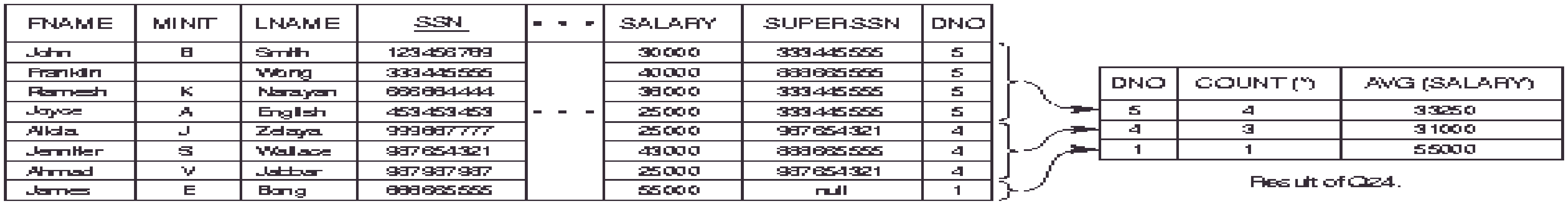
```
SELECT   DNO, COUNT (*), AVG (SALARY)
FROM     EMPLOYEE
GROUP BY   DNO;
```

- In Q24, the EMPLOYEE tuples are divided into groups--each group having the same value for the grouping attribute DNO.
- The COUNT and AVG functions are applied to each such group of tuples separately.
- The SELECT-clause includes only the grouping attribute and the functions to be applied on each group of tuples.

# 5.2 Grouping queries (3)

- Example (cont):

(a)



| FNAME | MINIT | LNAME | SSN | . . . | SALARY | SUPERSSN | DNO |
|---|---|---|---|---|---|---|---|
| John | B | Smith | 123456789 | | 30000 | 333445555 | 5 |
| Franklin | | Wong | 333445555 | | 40000 | 888665555 | 5 |
| Ramesh | K | Narayan | 666884444 | | 38000 | 333445555 | 5 |
| Joyce | A | English | 453453453 | . . . | 25000 | 333445555 | 5 |
| Alicia | J | Zelaya | 999887777 | | 25000 | 987654321 | 4 |
| Jennifer | S | Wallace | 987654321 | | 43000 | 888665555 | 4 |
| Ahmad | V | Jabbar | 987987987 | | 25000 | 987654321 | 4 |
| James | E | Borg | 888665555 | | 55000 | null | 1 |

| DNO | COUNT (*) | AVG (SALARY) |
|---|---|---|
| 5 | 4 | 33250 |
| 4 | 3 | 31000 |
| 1 | 1 | 55000 |

Result of Q24.

Grouping EMPLOYEE tuples by the value of DNO.

...

# 5.2 Grouping queries (4)

- A join condition can be used in conjunction with grouping.

- Example**:**

  Query 25. For each project, retrieve the project number, project name, and the number of employees who work on that project.

```
SELECT      PNUMBER, PNAME, COUNT (*)
FROM        PROJECT, WORKS_ON
WHERE       PNUMBER = PNO
GROUP BY    PNUMBER, PNAME;
```

  In this case, the grouping and aggregate functions are applied *after* the joining of the two relations.

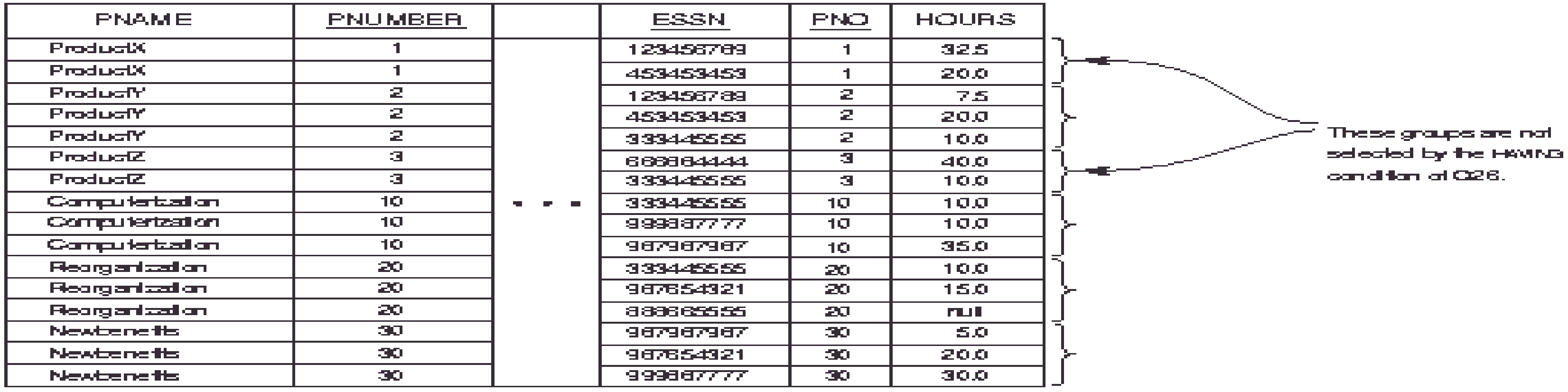
# 5.3 The HAVING clause (1)

- Sometimes we want to retrieve the values of aggregate functions for only those *groups that satisfy certain conditions.*
- The HAVING-clause is used for specifying a selection condition *on groups* (rather than *on individual tuples*)
- Example**:**
  Query 26. For each project *on which more than two employees work*, retrieve the project number, project name, and the number of employees who work on that project.

```
SELECT      PNUMBER, PNAME, COUNT (*)
FROM        PROJECT, WORKS_ON
WHERE       PNUMBER = PNO
GROUP BY    PNUMBER, PNAME
HAVING      COUNT (*) > 2;
```

# 5.3 The HAVING clause (2)

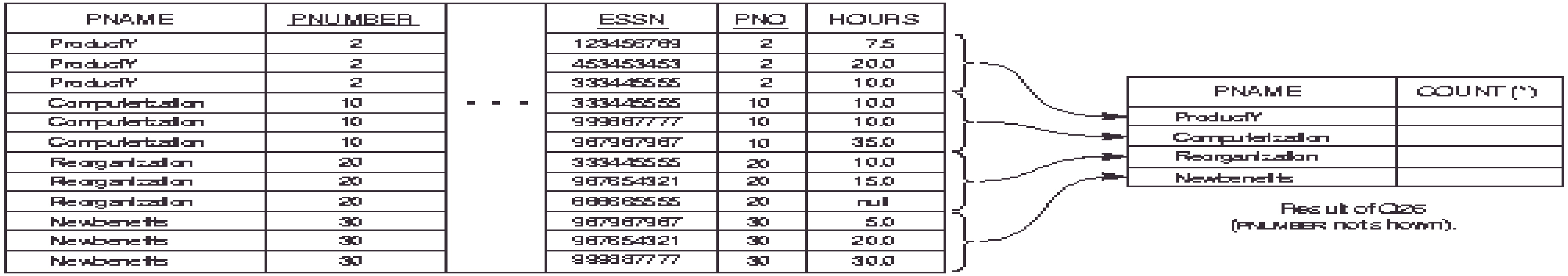
- Example (cont.):



| PNAME | PNUMBER |  | ESSN | PNO | HOURS |
|---|---|---|---|---|---|
| ProductX | 1 |  | 123456789 | 1 | 32.5 |
| ProductX | 1 |  | 453453453 | 1 | 20.0 |
| ProductY | 2 |  | 123456789 | 2 | 7.5 |
| ProductY | 2 |  | 453453453 | 2 | 20.0 |
| ProductY | 2 |  | 333445555 | 2 | 10.0 |
| ProductZ | 3 |  | 666884444 | 3 | 40.0 |
| ProductZ | 3 |  | 333445555 | 3 | 10.0 |
| Computerization | 10 | . . . | 333445555 | 10 | 10.0 |
| Computerization | 10 |  | 999887777 | 10 | 10.0 |
| Computerization | 10 |  | 987987987 | 10 | 35.0 |
| Reorganization | 20 |  | 333445555 | 20 | 10.0 |
| Reorganization | 20 |  | 987654321 | 20 | 15.0 |
| Reorganization | 20 |  | 888665555 | 20 | null |
| Newbenefits | 30 |  | 987987987 | 30 | 5.0 |
| Newbenefits | 30 |  | 987654321 | 30 | 20.0 |
| Newbenefits | 30 |  | 999887777 | 30 | 30.0 |

These groups are not selected by the HAVING condition of Q26.

After applying the WHERE clause but before applying HAVING.



| PNAME | PNUMBER |  | ESSN | PNO | HOURS |
|---|---|---|---|---|---|
| ProductY | 2 |  | 123456789 | 2 | 7.5 |
| ProductY | 2 |  | 453453453 | 2 | 20.0 |
| ProductY | 2 |  | 333445555 | 2 | 10.0 |
| Computerization | 10 | . . . | 333445555 | 10 | 10.0 |
| Computerization | 10 |  | 999887777 | 10 | 10.0 |
| Computerization | 10 |  | 987987987 | 10 | 35.0 |
| Reorganization | 20 |  | 333445555 | 20 | 10.0 |
| Reorganization | 20 |  | 987654321 | 20 | 15.0 |
| Reorganization | 20 |  | 888665555 | 20 | null |
| Newbenefits | 30 |  | 987987987 | 30 | 5.0 |
| Newbenefits | 30 |  | 987654321 | 30 | 20.0 |
| Newbenefits | 30 |  | 999887777 | 30 | 30.0 |

| PNAME | COUNT (*) |
|---|---|
| ProductY |  |
| Computerization |  |
| Reorganization |  |
| Newbenefits |  |

Result of Q26
(PNUMBER not shown).

After applying the HAVING clause condition.

# 5.3 The HAVING clause (3)

- Notice that, while *selection conditions in the* WHERE-clause *limit the tuples to which aggregate functions are applied*, the HAVING-clause serves to choose *whole groups*.

- Example**:**

  Query 27. For each project, retrieve the project number, project name, and the number of employees *from department 5* who work on that project.

```
SELECT      PNUMBER, PNAME, COUNT (*)
FROM        PROJECT, WORKS_ON, EMPLOYEE
WHERE      PNUMBER = PNO AND SSN =ESSN AND DNO = 5
GROUP BY   PNUMBER, PNAME;
```

  Here we restrict tuples in the relation (namely those that have DNO=5) *before the grouping*.

# 5.3 The HAVING clause (4)

- We have to be careful when two different conditions apply (one to the tuples in the relation and another to the aggregate function).

- Example**:**

  Query 28. Count the *total* number of employees whose salary exceed $40000 in each department, but only for departments where more than five employees work.

  The following query is INCORRECT (why?):

```
SELECT      DNAME, COUNT (*)
FROM        DEPARTMENT, EMPLOYEE
WHERE       DNUMBER = DNO AND SALARY > 40000
GROUP BY    DNAME
HAVING      COUNT(*) > 5;
```

# 5.3 The HAVING clause (5)

- Example (cont.):

  A correct formulation of the query:

```
SELECT     DNAME, COUNT (*)
FROM       DEPARTMENT, EMPLOYEE
WHERE      DNUMBER = DNO AND SALARY > 40000 AND
           DNO IN (SELECT     DNO
                   FROM       EMPLOYEE
                   GROUP BY   DNO
                   HAVING     COUNT(*) > 5)
GROUP BY   DNAME;
```

We assume that department names are unique (DNAME is a secondary key of DEPARTMENT).

# 6. Summary of SQL queries (1)

- A query in SQL can consist of up to six clauses, but only the first two, SELECT and FROM, are mandatory. The clauses are specified in the following order

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

- The SELECT-clause lists the attributes or functions to be retrieved
- The FROM-clause specifies all relations (or aliases) needed in the query but not those needed in nested queries
- The WHERE-clause specifies the conditions for selection and join of tuples from the relations specified in the FROM-clause
- GROUP BY specifies grouping attributes.
- HAVING specifies a condition for selection of groups
- ORDER BY specifies an order for displaying the result of a query.

# 6. Summary of SQL queries (2)

- A query is evaluated *conceptually* by applying (in this order)
  - the FROM clause (to compute the cross-product of the tables and joined tables involved in the query).
  - the WHERE clause (to delete rows in the cross-product that fail the conditions in the WHERE clause).
  - the GROUP BY clause (to group rows).
  - the HAVING clause (to eliminate groups that fail the conditions in the HAVING clause).
  - The SELECT clause (to eliminate columns that do not appear in the SELECT list, to eliminate duplicate rows if DISTINCT is specified, and to apply aggregate functions).
  - the ORDER BY clause (to sort the query result).

# 6. Summary of SQL queries (3)

- In general, the *conceptual evaluation* is not an efficient way of evaluating a query in a real system.
- A DBMS has special query optimization routines to decide on an *efficient query evaluation plan*.
- In general, there are *different ways* to specify the *same query* in SQL.
- It is the responsibility of the DBMS to execute a query efficiently independently of the way it is specified by the user.
- In practice, however, different specifications of the same query may result in internal evaluations plans with different cost.
- The user should be aware of which types of constructs in a query are more expensive to process than others.