



EMPLOYEE	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
John	B	Smith	10000739	1965-01-09	721 Foothills Drive, TX	M	8000	33344555	5	
Franco	T	Berry	30344555	1951-01-08	188 Pine Valley, TX	M	4000	88888888	5	
Allen	J	Zelba	88888777	1966-07-19	3551 Castle Springs, TX	F	5500	88764321	4	
Warren	S	Blalock	88764321	1947-08-05	271 Oak Bluffs, TX	F	4000	88888888	4	
Patterson	K	Namyan	66688444	1962-04-15	875 Fox Oak, Houston, TX	M	9000	33344555	5	
Chen	A	English	45678900	1970-07-01	3801 Hill, Houston, TX	F	2000	33344555	5	
Harold	J	Abner	88764321	1962-02-09	800 Oak, Houston, TX	M	5000	88764321	4	
Harvey	E	Berg	88888222	1957-11-10	400 Stone, Houston, TX	M	5000	NULL	1	

DEPT_LOCATIONS	DNUMBER	DLOCATION
1	Research	4
4	Stafford	5
5	Boston	5
5	Houston	5

DEPARTMENT	FNAME	DNUMBER	MGRSSN	MGRSTARTDATE
Research	Allen	30344555	1965-01-01	
Administration	Harold	88764321	1965-01-01	
Manufacturing	Warren	88888222	1967-08-10	

WORKS_ON	ESSN	PNO	HOURS
123456789	1	315	
123456789	2	715	
666666666	3	400	
456789000	1	200	
456789000	2	200	
303445555	2	100	
303445555	3	100	
303445555	10	100	
303445555	20	100	
888887777	30	200	
888887777	10	100	
888887777	10	100	
887643210	30	100	
887643210	30	200	
887643210	20	100	
888882220	20	100	

PROJECT	PNAME	DNUMBER	PLOCATION	DNUM
Project1	Project1	1	Boston	5
Project2	Project2	4	Stafford	5
Project3	Project3	5	Houston	5
Construction	Construction	10	Stafford	4
Manufacturing	Manufacturing	20	Houston	1
Manufacturing	Manufacturing	30	Stafford	4

DEPENDENT	ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP
303445555	Allen	Thompson	M	1965-01-05	CHILD
303445555	Jay		F	1968-09-05	SPOUSE
887643210	Abner		M	1962-02-09	SPOUSE
123456789	Harold		M	1962-01-04	SON
123456789	Anna		F	1968-03-26	CHILD
10456789	Ernest		F	1967-09-06	SPOUSE

Retrieval Queries in SQL

- SQL has one basic statement for retrieving information from a database; the SELECT statement
- Important distinction between SQL and the formal relational model; SQL allows a table (relation) to have two or more tuples that are identical in all their attribute values
- Hence, an SQL relation (table) is a *multi-set* (sometimes called a bag) of tuples; it is *not* a set of tuples
- SQL relations can be constrained to be sets by specifying PRIMARY KEY or UNIQUE attributes, or by using the DISTINCT option in a query

Simple SQL Queries

- Retrieve the birthdate and address of the employee whose name is 'John B. Smith'

```

SELECT BDATE, ADDRESS
FROM EMPLOYEE
WHERE FNAME='John' AND MINIT='B'
AND LNAME='Smith'
```

- SELECT-clause specifies the *attributes* and the WHERE-clause specifies the *selection condition*

Simple SQL Queries

- Retrieve the name and address of all employees who work for the 'Research' department

```

SELECT FNAME, LNAME, ADDRESS
FROM EMPLOYEE, DEPARTMENT
WHERE DNAME='Research' AND
DNUMBER=DNO
```

- Similar to a SELECT-PROJECT-JOIN sequence of relational algebra operations
- (DNUMBER=DNO) is a *join condition* (corresponds to a JOIN operation in relational algebra)

Simple SQL Queries

- Query 2: For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name, address, and birthdate.

```

Q2: SELECT PNUMBER, DNUMBER, LNAME, BDATE, ADDRESS
FROM PROJECT, DEPARTMENT, EMPLOYEE
WHERE DNUM=DNUMBER AND MGRSSN=SSN
AND PLOCATION='Stafford'
```

- In Q2, there are *two* join conditions
- The join condition DNUM=DNUMBER relates a project to its controlling department
- The join condition MGRSSN=SSN relates the controlling department to the employee who manages that department

Aliases

- In SQL, we can use the same name for two (or more) attributes as long as the attributes are in *different relations*
- A query that refers to two or more attributes with the same name must *qualify* the attribute name with the relation name by *prefixing* the relation name to the attribute name

Example:

```
EMPLOYEE.LNAME, DEPARTMENT.DNAME
```

- Some queries need to refer to the same relation twice
- In this case, *aliases* are given to the relation name

7

Aliases

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

```
Q8: SELECT E.FNAME, E.LNAME, S.FNAME,
        S.LNAME
        FROM EMPLOYEE E, EMPLOYEE S
        WHERE E.SUPERSSN=S.SSN
```

- In Q8, the alternate relation names E and S are called *aliases* or *tuple variables* for the EMPLOYEE relation
- We can think of E and S as two *different copies* of EMPLOYEE; E represents employees in role of *supervisees* and S represents employees in role of *supervisors*

8

Aliases

- Aliasing can also be used in any SQL query for convenience
- Can also use the AS keyword to specify aliases

```
Q8: SELECT E.FNAME, E.LNAME, S.FNAME,
        S.LNAME
        FROM EMPLOYEE AS E, EMPLOYEE AS S
        WHERE E.SUPERSSN=S.SSN
```

9

Unspecified Where clause

- A *missing WHERE-clause* indicates no condition; hence, *all tuples* of the relations in the FROM-clause are selected
 - This is equivalent to the condition WHERE TRUE
 - Query 9: Retrieve the SSN values for all employees
- ```
Q9: SELECT SSN
 FROM EMPLOYEE
```
- If more than one relation is specified in the FROM-clause *and* there is no join condition, then the *CARTESIAN PRODUCT* of tuples is selected

10

## Unspecified Where clause

- Example:

```
Q10: SELECT SSN, DNAME
 FROM EMPLOYEE, DEPARTMENT
```

- It is extremely important not to overlook specifying any selection and join conditions in the WHERE-clause; otherwise, incorrect and very large relations may result

11

## Use Of \*

- To retrieve all the attribute values of the selected tuples, a \* is used, which stands for *all the attributes*

### Examples:

```
Q1C: SELECT *
 FROM EMPLOYEE
 WHERE DNO=5
```

```
Q1D: SELECT *
 FROM EMPLOYEE, DEPARTMENT
 WHERE DNAME='Research' AND
 DNO=DNUMBER
```

12

## Use of Distinct

- SQL does not treat a relation as a set; *duplicate tuples can appear*
- To eliminate duplicate tuples in a query result, the keyword **DISTINCT** is used
- For example, the result of Q11 may have duplicate SALARY values whereas Q11A does not have any duplicate values

```

Q11: SELECT SALARY
 FROM EMPLOYEE
Q11A: SELECT DISTINCT SALARY
 FROM EMPLOYEE

```

13

## Set Operations

- SQL has directly incorporated some set operations
- There is a union operation (**UNION**), and in *some versions* of SQL there are set difference (**MINUS**) and intersection (**INTERSECT**) operations
- The resulting relations of these set operations are sets of tuples; *duplicate tuples are eliminated from the result*
- The set operations apply only to *union compatible relations*; the two relations must have the same attributes and the attributes must appear in the same order

14

## Set Operations

- 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.

```

Q4: (SELECT PNAME
 FROM PROJECT, DEPARTMENT, EMPLOYEE
 WHERE DNUM=DNUMBER AND MGRSSN=SSN
 AND LNAME='Smith')
 UNION
 (SELECT PNAME
 FROM PROJECT, WORKS_ON, EMPLOYEE
 WHERE PNUMBER=PNO AND ESSN=SSN AND
 LNAME='Smith')

```

15

## Nesting of Queries

- A complete SELECT query, called a *nested query*, can be specified within the WHERE-clause of another query, called the *outer query*
- Many of the previous queries can be specified in an alternative form using nesting
- Query 1: Retrieve the name and address of all employees who work for the 'Research' department.

```

Q1: SELECT FNAME, LNAME, ADDRESS
 FROM EMPLOYEE
 WHERE DNO IN (SELECT DNUMBER
 FROM DEPARTMENT
 WHERE DNAME='Research')

```

16

## Nesting of Queries

- The nested query selects the number of the 'Research' department
- The outer query select an EMPLOYEE tuple if its DNO value is in the result of either 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
- In general, we can have several levels of nested queries
- A reference to an *unqualified attribute* refers to the relation declared in the *innermost nested query*
- In this example, the nested query is *not correlated* with the outer query

17

## Correlated Nested Queries

- 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) the outer query*
- Query 12: Retrieve the name of each employee who has a dependent with the same first name as the employee.

```

Q12: SELECT E.FNAME, E.LNAME
 FROM EMPLOYEE AS E
 WHERE E.SSN IN (SELECT ESSN
 FROM DEPENDENT
 WHERE ESSN=E.SSN AND
 E.FNAME=DEPENDENT_NAME)

```

## Correlated Nested Queries

- In Q12, the nested query has a different result *for each tuple* in the outer query
- A query written with nested SELECT... FROM... WHERE... blocks and using the = or IN comparison operators can **always** be expressed as a single block query. For example, Q12 may be written as in Q12A

```
Q12A: SELECT E.FNAME, E.LNAME
 FROM EMPLOYEE E, DEPENDENT D
 WHERE E.SSN=D.ESSN AND
 E.FNAME=D.DEPENDENT_NAME
```

- The original SQL as specified for SYSTEM R also had a **CONTAINS** comparison operator, which is used in conjunction with nested correlated queries
- This operator was dropped from the language, possibly because of the difficulty in implementing it efficiently

19

## Correlated Nested Queries

- Most implementations of SQL *do not* have this operator
- The CONTAINS operator compares two *sets of values*, and returns TRUE if one set contains all values in the other set (reminiscent of the *division* operation of algebra).
  - Query 3: Retrieve the name of each employee who works on *all* the projects controlled by department number 5.

```
Q3: SELECT FNAME, LNAME
 FROM EMPLOYEE
 WHERE ((SELECT PNO
 FROM WORKS_ON
 WHERE SSN=ESSN)
 CONTAINS
 (SELECT PNUMBER
 FROM PROJECT
 WHERE DNUM=5))
```

20

## Correlated Nested Queries

- In Q3, the second nested query, which is not correlated with the outer query, retrieves the project numbers of all projects controlled by department 5
- The first nested query, which is correlated, retrieves the project numbers on which the employee works, which is different *for each employee tuple* because of the correlation

21

## The Exists Function

- EXISTS is used to check whether the result of a correlated nested query is empty (contains no tuples) or not
- We can formulate Query 12 in an alternative form that uses EXISTS as Q12B below

22

## The Exists Function

- Query 12: Retrieve the name of each employee who has a dependent with the same first name as the employee.

```
Q12B: SELECT FNAME, LNAME
 FROM EMPLOYEE
 WHERE EXISTS (SELECT *
 FROM DEPENDENT
 WHERE SSN=ESSN AND
 FNAME=DEPENDENT_NAME)
```

23

## The Exists Function

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

```
Q6: SELECT FNAME, LNAME
 FROM EMPLOYEE
 WHERE NOT EXISTS (SELECT *
 FROM DEPENDENT
 WHERE SSN=ESSN)
```

- In Q6, the correlated nested query retrieves all DEPENDENT tuples related to an EMPLOYEE tuple. If *none exist*, the EMPLOYEE tuple is selected
- EXISTS is necessary for the expressive power of SQL

24

## Explicit Sets

- It is also possible to use an **explicit (enumerated) set of values** in the WHERE-clause rather than a nested query
- Query 13: Retrieve the social security numbers of all employees who work on project number 1, 2, or 3.

```
Q13: SELECT DISTINCT ESSN
 FROMWORKS_ON
 WHERE PNO IN (1, 2, 3)
```

25

## NULLS in SQL Queries

- SQL allows queries that check if a value is NULL (missing or undefined or not applicable)
- SQL uses **IS** or **IS NOT** to compare NULLs because it considers each NULL value distinct from other NULL values, so equality comparison is not appropriate.
- Query 14: Retrieve the names of all employees who do not have supervisors.

```
Q14:SELECT FNAME, LNAME
 FROM EMPLOYEE
 WHERE SUPERSSN IS NULL
```

Note: If a join condition is specified, tuples with NULL values for the join attributes are not included in the result

26

## Joined Relations Feature in SQL2

- Can specify a "joined relation" in the FROM-clause
- Looks like any other relation but is the result of a join
- Allows the user to specify different types of joins (regular "theta" JOIN, NATURAL JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN, CROSS JOIN, etc)

27

## Joined Relations Feature in SQL2

- Examples:

```
Q8: SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME
 FROM EMPLOYEE E S
 WHERE E.SUPERSSN=S.SSN
```

can be written as:

```
Q8: SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME
 FROM (EMPLOYEE E LEFT OUTER JOIN EMPLOYEE S
 ON E.SUPERSSN=S.SSN)
```

```
Q1: SELECT FNAME, LNAME, ADDRESS
 FROM EMPLOYEE, DEPARTMENT
 WHERE DNAME='Research' AND DNUMBER=DNO
```

28

## Joined Relations Feature in SQL2

- could be written as:

```
Q1: SELECT FNAME, LNAME, ADDRESS
 FROM (EMPLOYEE JOIN DEPARTMENT
 ON DNUMBER=DNO)
 WHERE DNAME='Research'
```

or as:

```
Q1: SELECT FNAME, LNAME, ADDRESS
 FROM (EMPLOYEE NATURAL JOIN DEPARTMENT
 AS DEPT(DNAME, DNO, MSSN, MSDATE))
 WHERE DNAME='Research'
```

29

## Joined Relations Feature in SQL2

- Another Example;
  - Q2 could be written as follows; this illustrates multiple joins in the joined tables

```
Q2: SELECT PNUMBER, DNUM, LNAME,
 BDATE, ADDRESS
 FROM ((PROJECT JOIN DEPARTMENT ON
 DNUM=DNUMBER) JOIN EMPLOYEE
 ON MGRSSN=SSN)
 WHERE PLOCATION='Stafford'
```

30

## Aggregate Functions

- Include **COUNT**, **SUM**, **MAX**, **MIN**, and **AVG**
- Query 15: Find the maximum salary, the minimum salary, and the average salary among all employees.

```
Q15: SELECT MAX(SALARY),
 MIN(SALARY), AVG(SALARY)
 FROM EMPLOYEE
```

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

31

## Aggregate Functions

- Query 16: Find the maximum salary, the minimum salary, and the average salary among employees who work for the 'Research' department.

```
Q16: SELECT MAX(SALARY), MIN(SALARY),
 AVG(SALARY)
 FROM EMPLOYEE, DEPARTMENT
 WHERE DNO=DNUMBER AND
 DNAME='Research'
```

32

## Aggregate Functions

- Queries 17 and 18: Retrieve the total number of employees in the company (Q17), and the number of employees in the 'Research' department (Q18).

```
Q17: SELECT COUNT (*)
 FROM EMPLOYEE
```

```
Q18: SELECT COUNT (*)
 FROM EMPLOYEE, DEPARTMENT
 WHERE DNO=DNUMBER AND
 DNAME='Research'
```

33

## Grouping

- 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 the *grouping attribute(s)*
- The function is applied to each subgroup independently
- SQL has a **GROUP BY**-clause for specifying the grouping attributes, which *must also appear in the SELECT-clause*

34

## Grouping

- Query 20: For each department, retrieve the department number, the number of employees in the department, and their average salary.

```
Q20: SELECT DNO, COUNT (*), AVG (SALARY)
 FROM EMPLOYEE
 GROUP BY DNO
```

- In Q20, 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
- A join condition can be used in conjunction with grouping

35

## Grouping

- Query 21: For each project, retrieve the project number, project name, and the number of employees who work on that project.

```
Q21: SELECT PNUMBER, PNAME, COUNT (*)
 FROM PROJECT, WORKS_ON
 WHERE PNUMBER=PNO
 GROUP BY PNUMBER, PNAME
```

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

36

## The Having Clause

- Sometimes we want to retrieve the values of these 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)

37

## The Having Clause

- Query 22: 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.

```
Q22: SELECT PNUMBER, PNAME, COUNT(*)
 FROM PROJECT, WORKS_ON
 WHERE PNUMBER=PNO
 GROUP BY PNUMBER, PNAME
 HAVING COUNT (*) > 2
```

38

## Substring Comparison

- The **LIKE** comparison operator is used to compare partial strings
- Two reserved characters are used
  - '%' (or '\*' in some implementations)
    - replaces an arbitrary number of characters
  - '\_'
    - replaces a single arbitrary character

39

## Substring Comparison

- Query 25: Retrieve all employees whose address is in Houston, Texas. Here, the value of the ADDRESS attribute must contain the substring 'Houston,TX'.

```
Q25: SELECT FNAME, LNAME
 FROM EMPLOYEE
 WHERE ADDRESS LIKE
 '%Houston,TX%'
```

40

## Substring Comparison

- Query 26: Retrieve all employees who were born during the 1950s.
- Here, '5' must be the 8th character of the string (format of date DD-MON-YY), so the BDATE value is '\_\_\_\_5\_', with each underscore as a place holder for a single arbitrary character.

```
Q26: SELECT FNAME, LNAME
 FROM EMPLOYEE
 WHERE BDATE LIKE
 '____5_'
```

41

## Arithmetic Operators

- The standard arithmetic operators '+', '-', '\*', and '/' can be applied to numeric values in an SQL query result
- Query 27: Show the effect of giving all employees who work on the 'ProductX' project a 10% raise.

```
Q27: SELECT FNAME, LNAME, 1.1*SALARY
 FROM EMPLOYEE, WORKS_ON, PROJECT
 WHERE SSN=ESSN AND PNO=PNUMBER AND
 PNAME='ProductX'
```

42

## Order By

- The **ORDER BY** clause is used to sort the tuples in a query result based on the values of some attribute(s)
- Query 28: Retrieve a list of employees and the projects each works in, ordered by the employee's department, and within each department ordered alphabetically by employee last name.

```
Q28: SELECT DNAME, LNAME, FNAME, PNAME
 FROM DEPARTMENT, EMPLOYEE,
 WORKS_ON, PROJECT
 WHERE DNUMBER=DNO AND SSN=ESSN
 AND PNO=PNUMBER
 ORDER BY DNAME, LNAME
```

43

## Order By

- The default order is in ascending order of values
- Specify the keyword **DESC** if we want a descending order
  - ORDER BY DNAME DESC
- The keyword **ASC** can be used to explicitly specify ascending order, even though it is the default

44

## Summary of SQL Queries

- 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>]
```

45

## Summary of SQL Queries

- 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
- A query is evaluated by first applying the WHERE-clause, then GROUP BY and HAVING, and finally the SELECT-clause

46

## Specifying Updates in SQL

- SQL commands to modify the database
  - INSERT
    - Add tuples to a relation
    - Attribute values should be in same order as create table command
  - DELETE
    - Removes tuples from a relation
  - UPDATE
    - modify attribute values of one or more selected tuples

47

## Insert

- Example:

```
U1: INSERT INTO EMPLOYEE
 VALUES ('Richard','K','Marini', '653298653', '30-DEC-52',
 '98 Oak Forest,Katy,TX', 'M', 37000,'987654321', 4)
```

- An alternate form of INSERT specifies explicitly the attribute names that correspond to the values in the new tuple
- Attributes with NULL values can be left out
- Example: Insert a tuple for a new EMPLOYEE for whom we only know the FNAME, LNAME, and SSN attributes.

```
U1A: INSERT INTO EMPLOYEE (FNAME, LNAME, SSN)
 VALUES ('Richard', 'Marini', '653298653')
```

48

## Insert

- Example: Suppose we want to create a temporary table that has the name, number of employees, and total salaries for each department. A table DEPTS\_INFO is created by U3A, and is loaded with the summary information retrieved from the database by the query in U3B.

```
U3A: CREATE TABLE DEPTS_INFO
 (DEPT_NAME VARCHAR(10),
 NO_OF_EMPS INTEGER,
 TOTAL_SAL INTEGER);

U3B: INSERT INTO DEPTS_INFO (DEPT_NAME,
 NO_OF_EMPS, TOTAL_SAL)
 SELECT DNAME, COUNT (*), SUM (SALARY)
 FROM DEPARTMENT, EMPLOYEE
 WHERE DNUMBER=DNO
 GROUP BY DNAME ;
```

49

## Delete

- Includes a WHERE-clause to select the tuples to be deleted
- Tuples are deleted from only *one table* at a time (unless CASCADE is specified on a referential integrity constraint)
- A missing WHERE-clause specifies that *all tuples* in the relation are to be deleted; the table then becomes an empty table
- The number of tuples deleted depends on the number of tuples in the relation that satisfy the WHERE-clause
- Referential integrity should be enforced

50

## Delete

- Examples:

```
U4A: DELETE FROM EMPLOYEE
 WHERE LNAME='Brown'

U4B: DELETE FROM EMPLOYEE
 WHERE SSN='123456789'

U4C: DELETE FROM EMPLOYEE
 WHERE DNO IN
 (SELECT DNUMBER
 FROM DEPARTMENT
 WHERE DNAME='Research')

U4D: DELETE FROM EMPLOYEE
```

51

## Update

- Used to modify attribute values of one or more selected tuples
- A WHERE-clause selects the tuples to be modified
- An additional SET-clause specifies the attributes to be modified and their new values
- Each command modifies tuples *in the same relation*
- Referential integrity should be enforced

52

## Update

- Example: give all employees in the research department a 10% raise.

```
UPDATE EMPLOYEE
SET SALARY=SALARY*1.1
WHERE DNO IN (SELECT DNUMBER
 FROM DEPARTMENT
 WHERE DNAME='Research');
```

53

## Exercises

- List the names of employees who have at least one dependent.

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

54

## Exercises

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

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

55

## Exercises

- Retrieve the names of all employees who have two or more dependents

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

56

## Exercises

- Retrieve the names of all employees whose salary is greater than the salary of all the employees in department 5.

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

57