SQL: SELECT (cont.)

- **Example 1**: find the name and address of employees working in the 'Research' department

```
SELECT fname, lname, address
FROM Employee, Departemnt
WHERE dno = dnumber AND dname = 'Research'
```

- The following is the equivalence query in Relational Algebra:

  - $\pi_{fname,\ lname,\ address}\ (\ \sigma_{(dname='Research'\ and\ dnumber\ =\ dno)}\ (\ employee \times department\ )\ \ )$

SQL: SELECT (cont.)

- **Example 2**: For the projects located in 'Stafford', find the name of the project, the name of the controlling department, the last name of the department's manager, his address and birthdate

Projects in 'Stafford' can be found as follows:

```
SELECT pnumber, pname, plocation
FROM project
WHERE plocation = 'Stafford';
```

The controlling department of these projects is added as follows:

```
SELECT pnumber, pname, dname, mgrssn
FROM project, department
WHERE dnum = dnumber AND plocation = 'Stafford';
```

Finally, the manager information is added as follows:

```
SELECT pname, dname, lname, address bdate
FROM project, department, employee
WHERE dnum = dnumber AND mgrssn = ssn AND plocation = 'Stafford';
```

Qualifying Attribute with Relation Names and Aliasing

- Ambiguous attribute names:
  - **Attributes** in different relations can have the same attribute name.
- Example:

```
Works_on (Essn, Pno, Hours);
Dependent (Essn, Dependent_name, Sex, Bdate, Relationship);
```

- When **Works_on** and **Dependent** relations appear in the same query, the attribute name **Essn** will be ambiguous
  - Find project numbers of projects worked on by employees who has a daughter names ‘Alice’

```
SELECT Pno
FROM Works_on, Dependent
WHERE Essn = Essn AND name = ‘Alice’
```

(You cannot tell that Essn is an attribute from the **Project** relation or from the **Dependent** relation

Solving the Ambiguous Attribute Names from Different Relations

- When different relations has a common attribute name, we can make that name un-ambiguous by:
  - Qualifying (= prefixing) the **attribute name** with the source relation name.

```
Project.essn        refers the the essn attribute in Project relation
Dependent.essn      refers the the essn attribute in Dependent relation
```

  - Example:

> ▪ Find **project numbers** of **projects** worked on by **employees** who has a **daughter** named **'Alice'**.

```
SELECT pno
FROM   works_on, dependent
WHERE  works_on.essn = dependent.essn
  AND  name = 'Alice';
```

Aliasing: qualifying attribute names from the same relation

- Fact:
  - Sometimes, we need to use the **same relation** multiple times in a **SELECT** command
  - Every attribute name of that **relation** will be ambiguous
- **Example**: for each employee, retrieve the employee's first and last name and the first and last name of his or her immediate supervisor.

```
SELECT   Employee.Fname, Employee.Lname, Employee.Fname,
Employee.Lname
FROM     Employee, Employee
WHERE    Employee.Super_ssn = Employee.Ssn;
```

Aliasing: qualifying attribute names from the same relation (cont.)

- **Alias**: solving ambiguous name from the same relation

- 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.superssn = S.ssn;
```

This practice is recommended since it results in queries that are easier to comprehend.

Renaming the Output Attribute Name

- Consider the output

```
SELECT e.fname, e.lname,
FROM   employee e, emplo
WHERE  e.superssn = m.ss

Output:

+--------+---------+
| fname  | lname   |
+--------+---------+
| John   | Smith   |
| Ramesh | Narayan |
| Joyce  | English |
| Alicia | Zelaya  |
| Ahmad  | Jabbar  |
| Frankl | Wong    |
| Jennif | Wallace |
| James  | Borg    |
+--------+---------+
```

```
SELECT e.fname EmpFN, e.lname EmpLN, m.fname SupFN, m.lname SupFN
FROM   employee e, employee m
WHERE  e.superssn = m.ssn;

Output:

+--------+---------+--------+---------+
| EmpFN  | EmpLN   | SupFN  | SupFN   |
+--------+---------+--------+---------+
| John   | Smith   | Frankl | Wong    |
| Ramesh | Narayan | Frankl | Wong    |
| Joyce  | English | Frankl | Wong    |
| Alicia | Zelaya  | Jennif | Wallace |
| Ahmad  | Jabbar  | Jennif | Wallace |
| Frankl | Wong    | James  | Borg    |
| Jennif | Wallace | James  | Borg    |
+--------+---------+--------+---------+
```

The **name (title)** of the **attributes** are not very **meaningful**

The * Selector

- The **SELECT *** selects all attributes in all relations in the FROM clause

```
Q1C:    SELECT    *
        FROM      EMPLOYEE
        WHERE     Dno = 5;

Q1D:    SELECT    *
        FROM      EMPLOYEE, DEPARTMENT
        WHERE     Dname = 'Research' AND Dno = Dnumber;

Q10A:   SELECT    *
        FROM      EMPLOYEE, DEPARTMENT;
```

Tables as Sets in SQL

- SQL usually treats a table **not** as a set:
  - **Duplicate tuples** can appear more than once in a table, and in the result of a query
  - But an SQL table with a key is restricted to being a set
  - Use the **DISTINCT** keyword in the SELECT clause to remove duplicate values
  - Retrieve the salary of **every** employee and all **distinct** salary values

```
Q11:   SELECT   ALL Salary
       FROM     EMPLOYEE;

Q11A:  SELECT   DISTINCT Salary
       FROM     EMPLOYEE;
```

(a)

Salary
30000
40000
25000
43000
38000
25000
25000
55000

(b)

Salary
30000
40000
25000
43000
38000
55000

Tables as Sets in SQL (cont.)

- R has the following set operators

```
(SELECT pname
FROM   project, works_on, employee
WHERE  pnumber = pno
  AND  essn = ssn
  AND  lname = 'Smith')
UNION
(SELECT pnumber
FROM   project, works_on, employee
WHERE  pnumber = pno
  AND  essn = ssn
  AND  lname = 'Borg');
```

```
+----------+
| pname    |
+----------+
| ProductX |
| ProductY |
| 20       |  <----- How is 20 a pname ????
+----------+
```

- Example: MySQL (version 5.5.24) will even accept the following UNION:
  - MySQL do not know about the meaning of each attribute

Other Tuple Conditions

- We have seen that you can use "ordinary" Boolean conditions in the where clause
- Other tuple conditions
  - The membership test operator (**IN** and **NOT IN**)
  - The any selection operator (**ANY**)
  - The all selection operator (**ALL**)
  - The non-empty set test operator (**EXISTS**)
  - The null test operator (**IS NULL**)
  - Wild card characters (_ and %)

The IN and NOT-IN Comparison Operator

- The comparison operator **IN** tests whether a value is contained in a set
  - Syntax: attr **IN** (set of values)
- Example:
  - Find the fname and lname of employees whose SSN is 123456789 or 333445555
  - Find the fname and lname of employees whose SSN is not equal to 123456789 or 333445555

```
SELECT fname, lname
FROM   employee
WHERE  ssn IN ('123456789', '333445555');
```

```
SELECT fname, lname
FROM   employee
WHERE  ssn NOT IN ('123456789', '333445555');
```

The ANY Selection Operator

- ANT
  - Syntax: attr RelationalOperator **any** (set of values)
  - Meaning:

> - The **(Boolean) expression** evaluates to **true** if the expression `attr RelationalOperator x` is **true** for ***at least one*** **member** of the `( set of values )`
> - The **(Boolean) expression** evaluates to **false** ***otherwise***

- Example:

```
SELECT fname, lname
FROM   employee
WHERE  salary >= ANY ( 30000, 50000 )
```

Note: why is the above quey illegal?

- In the SQL syntax, you cannot use an **explicit set of value**
- Instead, you must use a **subquery** that returns **a set of values**

The ALL Selection Operator

- All
  - Syntax: attr RelationalOperator **all** (set of values)
  - Meaning:

    - The **(Boolean) expression** evaluates to **true** if the expression `attr RelationalOperator x` is **true** for ***all*** **members** of the `( set of values )`
    - The **(Boolean) expression** evaluates to **false** ***otherwise***

- Example

```
SELECT fname, lname
FROM   employee
WHERE  salary >= ALL ( 30000, 50000 )
```

The ! = ALL Selection Operator

- Example: **! = ALL**

```
SELECT fname, lname
FROM   employee
WHERE  ssn != ALL ('111-11-1111', '222-22-2222’)
```

- x **!= ALL** (set of values) is the same as x **NOT IN** (set of values)
- x = **ANY** (set of values) is the same as x **IN** (set of values)

Relations of One Tuple

- Example:
  - Find employees who salary is greater than or equal to John Smith

```
SELECT fname, lname, salary, dno
from employee A
where salary >= (select salary
                 from employee
                 where fname='John' and lname='Smith')
```

Because there is only one salary value for 'John Smith', we can leave of ALL or ANY in the set clause.

  - Another one

```
SELECT fname, lname, salary, dno
from employee A
where salary >= (select salary
                 from employee
                 where dno = 5)
```

If you omit ALL or ANY keywords on a set of size ≥ 2, SQL will report an error !!!

Because this query returns **more** than 1 tuple !

You get this error: Subquery returns more than 1 row

The EXISTs and IS NULL

- Exists (set of values)
  - Example:

    exists ('a', 'b') // returns true

    exists ( ) // returns false

- IS NULL: testing for null values
  - Find all employee tuples that has a NULL value in the salary attribute:

```
SELECT *
FROM   employee
WHERE  salary IS NULL
```

LIKE: Wildcard String Comparison

- Example: Find fname and lname of employees whose last name start with an ‘S’?
- When comparing strings, you can use the **LIKE** operator to perform wildcard string comparison
  - Example: Find fname and lname of employees whose last name start with an 'S’
  - Solution:

```
SELECT fname, lname
FROM   E.mployee
WHERE  lname LIKE 'S%'
```

Percent (%) matches 0 or more characters

LIKE: Wildcard String Comparison (cont.)

- Example: Find all employees who were born during the 1970s
  - Solution:

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

Underscore (_) matches exactly one character

- **Wildcards** can ONLY be used with the **LIKE** operator
  - The following query will find people with last name equal to 'S%':

```
SELECT fname, lname
FROM   employee
WHERE  lname = 'S%'
```

More Tuple Conditions

- Another feature allows the use of **arithmetic** in queries
  - +, –, *, and division /

- **Example**: Show the resulting salaries if every employee working on the ‘ProductX’ project is given a 10% raise.

```
SELECT E.Fname, E.Lname, 1.1 * E.Salary AS Increased_sal
FROM EMPLOYEE AS E, WORKS_ON AS W, PROJECT AS P
WHERE E.Ssn = W.Essn AND W.Pno = P.Pnumber AND P.Pname = ‘ProductX’;
```

Nested Queries

- Nested query
  - A **nested query** is a SELECT query where the WHERE clause contains subquery.

```
SELECT ...
FROM   ...
WHERE  ... (SELECT ...     /* Subquery */
            FROM ...
            WHERE ...)
```

  - In theory, nesting can be arbitrarily deep

Examples of simple (= non-correlated) nested queries

- Query 1:
  - Find fname, lname of **employees** in the "Research" **department**
  - Solution: using a NON-nested query (uses a **join**):

```
SELECT fname, lname
FROM   employee, department
WHERE  dno = dnumber AND  dname = 'Research'
```

  - Solution using a **nested query**

Examples of simple (= non-correlated) nested queries (cont.)

- Suppose we know that the 'Research' department has the department number 5, we can write the query as:

```
SELECT fname, lname
FROM   employee
WHERE  dno = 5
```

- We can use a **subquery** to find the department number of the Research department:
  - (SELECT dnumber FROM department WHERE dname = 'Research')
- Therefore, th **ery** as follows:

```
SELECT fname, lname
FROM   employee
WHERE  dno IN
              ( SELECT pnumber
                FROM department
                WHERE dname = 'Research' )
```

Examples of simple (= non-correlated) nested queries (cont.)

- Query 2:
  - Find fname, lname of **employees** that do not have any dependent
  - 1) Conceptual solution

```
SELECT fname, lname
FROM   employee
WHERE  ssn IN
          { ssn of employees without dependent }
```

  - 2) Which is equivalent to:

```
SELECT fname, lname
FROM   employee
WHERE  ssn NOT IN
          { ssn of employees with (one or more) dependent }
```

  - 3) We can write this in SQL as:

```
SELECT fname, lname
FROM   employee
WHERE  ssn NOT IN
          (SELECT essn
           FROM   dependent);
```

Correlated Nested Queries

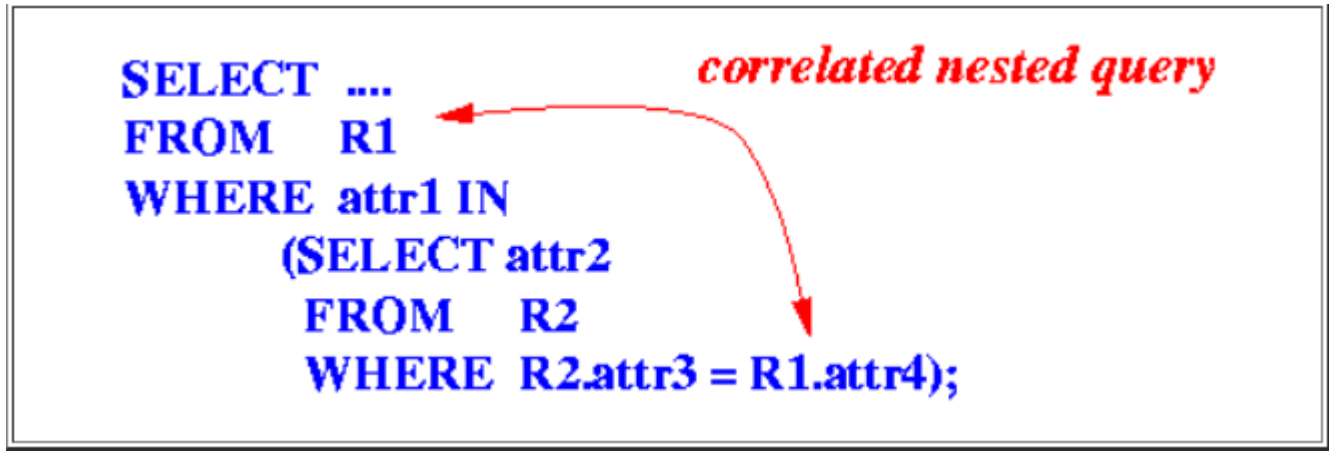
- So far, the nested queries above are uncorrelated:
  - The inner (sub)query is a **stand-alone** query that can be executed independently from the **outer query**
  - Example:

```
SELECT fname, lname
FROM   employee
WHERE  ssn NOT IN
           (SELECT essn            /* Can be executed by itself */
            FROM   dependent)
```

  - The inner query is completely independent from the outer query.

Correlated Nested Queries (cont.)

- Correlated nested query
  - A query where the inner query (i.e., the query in the WHERE clause) uses one or more attributes from relation(s) specified in the outer query.



  - The attribute attr4 of the relation R1 in the outer query is used in the inner query

Meaning of Ne

- Consider the following ex

```
SELECT fname, lname, salary, dr
FROM employee A
WHEre salary >= ALL (select sal
                     from emp
                     where B.d
```

- Suppose the employee r following tuples:

```
Outer tuple: A = John   Smith    30000.00  5

      WHERE salary   >= ALL (select salary from employee B where B.dno=A.dno)
  ==> WHERE 30000.00 >= ALL (select salary from employee B where B.dno=5)
  ==> WHERE 30000.00 >= ALL (30000, 40000, 38000, 25000)
  ==> FALSE

Outer tuple: A = Frankl Wong     40000.00  5

      WHERE salary   >= ALL (select salary from employee B where B.dno=A.dno)
  ==> WHERE 40000.00 >= ALL (select salary from employee B where B.dno=5)
  ==> WHERE 40000.00 >= ALL (30000, 40000, 38000, 25000)
  ==> TRUE  (select outer tuple !!!)

Outer tuple: A = Alicia Zelaya   25000.00  4

      WHERE salary   >= ALL (select salary from employee B where B.dno=A.dno)
  ==> WHERE 25000.00 >= ALL (select salary from employee B where B.dno=4)
  ==> WHERE 25000.00 >= ALL (25000, 43000)
  ==> FALSE
```

fname	lname	salary	dno
John	Smith	30000.00	5
Frankl	Wong	40000.00	5
Alicia	Zelaya	25000.00	4
Jennif	Wallace	43000.00	4
Ramesh	Narayan	38000.00	5
Joyce	English	25000.00	5
Ahmad	Jabbar	25000.00	4
James	Borg	55000.00	1

Output:

fname	lname	salary	dno
Frankl	Wong	40000.00	5
Jennif	Wallace	43000.00	4
James	Borg	55000.00	1

Examples on How to Use Correlated Nested Queries

- Example 1: Find fname, lname of **employees** that do not have any **dependent**

```
SELECT fname, lname
FROM Employee
WHERE { set of dependents of this employee }
          = empty set
```

```
SELECT fname, lname
FROM employee
WHERE (  select *
            from dependent
            where essn = employee.ssn)
          = empty set
```

```
SELECT fname, lname
FROM  employee
WHERE not exists (select *
                        from dependent
                        where essn = employee.ssn)
```

Examples on how to use correlated nested queries (cont.)

- Example 2: Find fname, lname of **employees** that do not work on any **project** controlled by the **Research department**

```
SELECT fname, lname
FROM Employee A
WHERE {  set of projects w.o. by employee A
            and controlled by Research dept }
         = empty set
```

```
SELECT fname, lname
FROM Employee A
WHERE not exists
      { set of projects w.o. by employee A
          and controlled by Research dept }
```

```
SELECT fname, lname
FROM employee A
WHERE not exists
        ( select *
          from    projects
          where  project w.o. by employee A
                and project controlled by Research dept )
```

```
SELECT fname, lname
FROM employee A
WHERE not exists
          (select *
           from   projects
           where  pnumber in ( project w.o. by employee A )
      and pnumber in ( project controlled by Research dept )
)
```

Examples on how to use correlated nested queries (cont.)

- Example 2 (cont.): Find fname, lname of **employees** that do not work on any **project** controlled by the **Research department**

```
SELECT fname, lname
FROM employee A
WHERE not exists
          (select *
           from   project
           where  pnumber in (select pno
                              from works_on
                              where  essn = A.ssn )
           and
                    pnumber in (select pnumber
                                from   project, department
                                where  dnum = dnumber and  dname = 'Research' ) )
```

Ordering of Query Results

- ORDER BY clause
  - Order the tuples in the result of a query by the values of one or more of the attributes that appear in the query result
- Example:
  - Retrieve a list of **employees** and the **projects** they are working on, ordered by department and, within each department, ordered alphabetically by last name, then first name.

```
SELECT D.Dname, E.Lname, E.Fname, P.Pname
FROM   DEPARTMENT AS D, EMPLOYEE AS E, WORKS_ON AS W, PROJECT AS P
WHERE  D.Dnumber = E.Dno AND E.Ssn = W.Essn AND W.Pno = P.Pnumber
ORDER BY D.Dname, E.Lname, E.Fname;
```

INSERT, DELETE, and UPDATE Statements in SQL

- The insert command
  - The INSERT command can be used to insert one or more **tuples** into a **relation**
- Format 1: inserting one complete tuple:
  - Must specify all attribute values in the exact order at the **relation schema**

```
INSERT INTO     EMPLOYEE
VALUES          ( 'Richard', 'K', 'Marini', '653298653', '1962-12-30', '98
                Oak Forest, Katy, TX', 'M', 37000, '653298653', 4 );
```

- Format 2: inserting a partial tuple
  - Specify a subset of the attribute values in the any order

```
INSERT INTO employee(fname, lname, ssn)
VALUES ('Richard', 'Marini', '222669999');
```

NOTE: The attributes not in the list are given their DEFAULT or NULL value

INSERT Command in SQL

- DBMS should enforce all the integrity constraints due to insert command
  - How about the following insert operation

INSERT INTO EMPLOYEE (Fname, Lname, Ssn, Dno)
VALUES ('Robert', 'Hatcher', '980760540', 2);

(U2 is rejected if referential integrity checking is provided by DBMS.)

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

INSERT INTO EMPLOYEE (Fname, Lname, Dno)
VALUES ('Robert', 'Hatcher', 5);

(U2A is rejected if NOT NULL checking is provided by DBMS.)

INSERT Command in SQL

- Inserting a tuple using the result from a SELECT command
  - Syntax: INSERT INTO relationName (SELECT ...)
- Example:
  - To create a temporary table that has the employee last name, project name, and hours per week for each employee working on a project

```
CREATE TABLE WORKS_ON_INFO
    ( Emp_name VARCHAR(15), Proj_name VARCHAR(15), Hours_per_week DECIMAL(3,1) );

INSERT INTO WORKS_ON_INFO ( Emp_name, Proj_name, Hours_per_week )
    SELECT    E.Lname, P.Pname, W.Hours
    FROM     PROJECT P, WORKS_ON W, EMPLOYEE E
    WHERE    P.Pnumber = W.Pno AND W.Essn = E.Ssn;
```

Need to keep a view to keep table up to data

DELETE Command in SQL

- Delete one or more tuples from a relation using the DELETE command.
  - Syntax:

```
DELETE FROM relationName
WHERE  tuple-boolean-condition
```

  - Example 1: Delete all **employees** from the 'Research' department

```
DELETE FROM Employee
WHERE  dno  IN (SELECT dnumber
                FROM   department
                WHERE  dname = 'Research')
```

DELETE Command in SQL (cont.)

- Example 3:
  - Delete all employees from the 'Research' department have more than 2 dependents:

```
DELETE FROM employee
WHERE  dno  IN ( SELECT dnumber
                 FROM   department
                 WHERE  dname = 'Research')
    AND  ssn  IN (  SELECT   essn
                    FROM     dependent
                    GROUP BY essn
                    HAVING   COUNT(name) > 2)
```

UPDATE Command in SQL

- Update one attribute value in one or more tuples from a relation using the UPDATE command
  - Syntax:

```
UPDATE relationName
SET    attributeName = expression
WHERE  tuple-boolean-condition
```

  - Example1: Change the address of employee 'John Smith' to '123 Pike Lane, Austin, TX’

```
UPDATE employee
SET    address = '123 Pike Lane, Austin, TX'
WHERE  fname = 'John’ AND  lname = 'Smith';
```

UPDATE Command in SQL (cont.)

- Example2: Give all employees in the 'Research' department a 10% raise

```
UPDATE   Employee
SET      salary = 1.1 * salary
WHERE    dno IN ( SELECT dnumber
                  FROM   department
                  WHERE  dname = 'Research')
```