Introduction to SQL: Data Retrieving

Ruslan Fomkin

Databasdesign för Ingenjörer – 1056F

Structured Query Language (SQL)

- History:
  - SEQUEL (Structured English QUery Language), earlier 70's, IBM Research
  - SQL (ANSI 1986), SQL1 or SQL-86
  - SQL2 (SQL-92)
  - SQL-99 (SQL3)
    - core specification and optional specialized packages
- Standard language for commercial DBMS
  - each DBMS has own features over standard

SQL includes

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
  - Queries
  - Updates
- Additional facilities
  - views
  - security and authorization
  - integrity constraints
  - transaction controls
  - rules for embedding SQL statements into, e.g., Java, C++

SQL based on

- Formal Relational Data Model
  - Terminology

    relation - table

    tuple - row

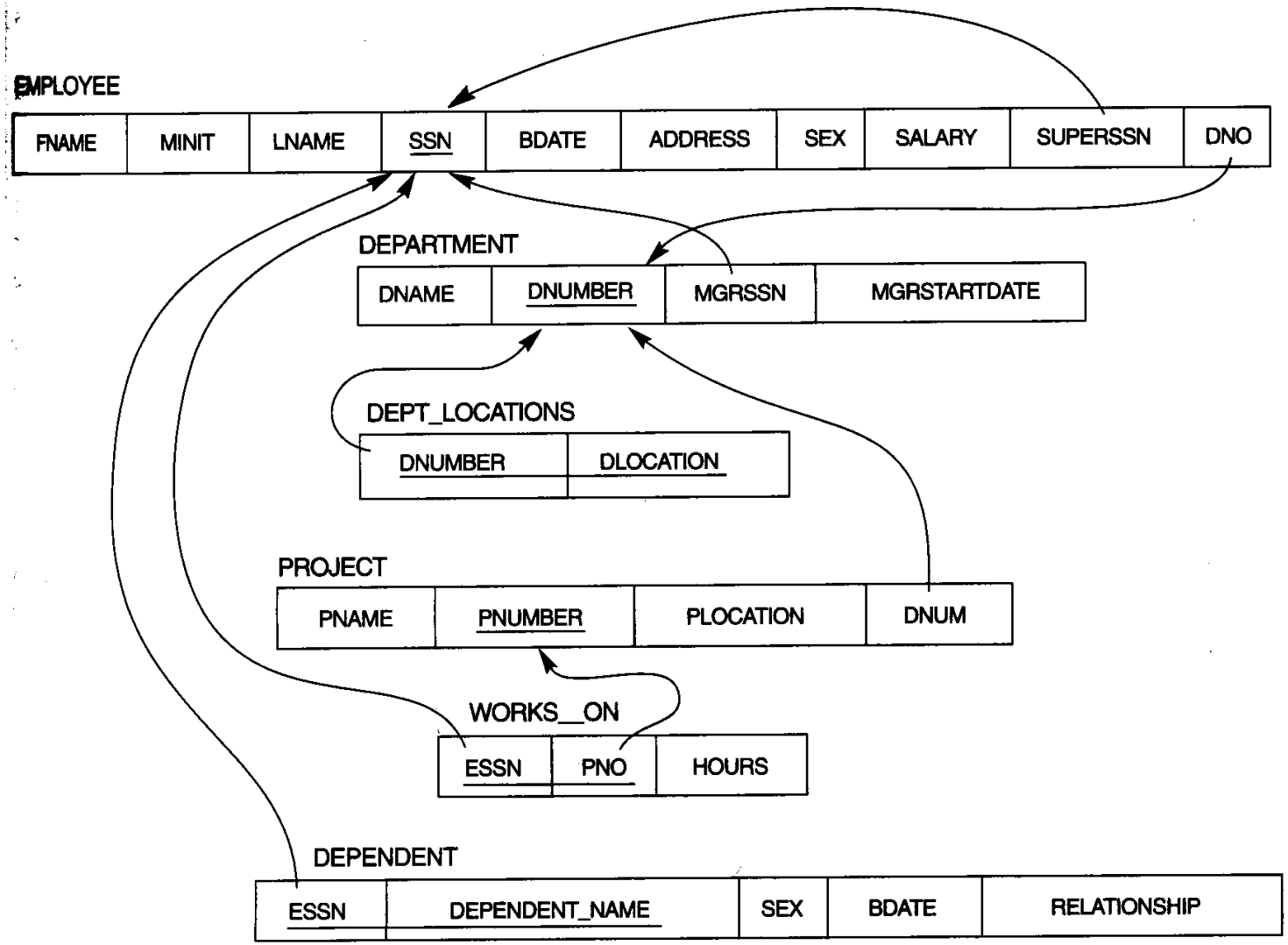
    attribute - column
  - SQL allows a table to have duplicates
- Tuple Relational Calculus
  - Includes some operations from relational algebra

Basic query statement of SQL

SELECT $A_1, A_2, \ldots, A_n$
FROM $r_1, r_2, \ldots, r_m$
WHERE P

- $A_1, A_2, \ldots, A_n$ – list of the attribute names whose values to be retrieved be the query
- $r_1, r_2, \ldots, r_m$ – list of the table names required to process the query
- P – conditional expression that identifies the tuples to be retrieved by the query
  - connectors: **AND**, **OR**, **NOT**
  - comparison operations: =, <, <=, >, >=, <>
- Result of the query is a table

Example database (from E/N ch. 5)



Example data

EMPLOYEE	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
	John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
	Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
	Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
	Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
	Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
	Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
	Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
	James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	null	1

DEPT_LOCATIONS	DNUMBER	DLOCATION
	1	Houston
	4	Stafford
	5	Bellaire
	5	Sugarland
		Houston

DEPARTMENT	DNAME	DNUMBER	MGRSSN	MGRSTARTDATE
	Research	5	333445555	1988-05-22
	Administration	4	987654321	1995-01-01
	Headquarters	1	888665555	1981-06-19

WORKS_ON	ESSN	PNO	HOURS
	123456789	1	32.5
	123456789	2	7.5
	666884444	3	40.0
	453453453	1	20.0
	453453453	2	20.0
	333445555	2	10.0
	333445555	3	10.0
	333445555	10	10.0
	333445555	20	10.0
	999887777	30	30.0
	999887777	10	10.0
	987987987	10	35.0
	987987987	30	5.0
	987654321	30	20.0
	987654321	20	15.0
	888665555	20	null

PROJECT	PNAME	PNUMBER	PLOCATION	DNUM
	ProductX	1	Bellaire	5
	ProductY	2	Sugarland	5
	ProductZ	3	Houston	5
	Computerization	10	Stafford	4
	Reorganization	20	Houston	1
	Newbenefits	30	Stafford	4

DEPENDENT	ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP
	333445555	Alice	F	1986-04-05	DAUGHTER
	333445555	Theodore	M	1983-10-25	SON
	333445555	Joy	F	1958-05-03	SPOUSE
	987654321	Abner	M	1942-02-28	SPOUSE
	123456789	Michael	M	1988-01-04	SON
	123456789	Alice	F	1988-12-30	DAUGHTER
	123456789	Elizabeth	F	1967-05-05	SPOUSE

Query 0 (simple query)

- Retrieve the birthdate and address of the employee(s) whose last name is ‘Smith’

```
SELECT   BDATE, ADDRESS
FROM     EMPLOYEE
WHERE    LNAME=‘Smith’;
```

- Result

```
BDATE       ADDRESS
========== ==============================
1965-01-09 731 Fondren, Houston, TX
```

Query 1 (select-project-join query)

- 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;

- Result

```
FNAME           LNAME           ADDRESS
=============== =============== ========================
John            Smith           731 Fondren, Houston, TX
Franklin        Wong            638 Voss, Houston, TX
Joyce           English         5631 Rice, Houstan, TX
Ramesh          Narayan         975 Fire Oak, Humble, TX
```

Query 2 (more complex query)

- For every project located in ‘Stafford’, list the project number, the controlling department number, and the department manager’s last name, address, and birthdate

SELECT PNUMBER, DNUM, LNAME, ADDRESS, BDATE

FROM PROJECT, DEPARTMENT, EMPLOYEE

WHERE DNUM=DNUMBER **AND** MGRSSN=SSN **AND** PLOCATION=‘Stafford’;

- Result

```
PNUMBER  DNUM  LNAME    ADDRESS                      BDATE
======== ===== ======== ============================ =========
      10     4 Wallance 291 Berry, Bellaire, TX 1941-06-20
      30     4 Wallance 291 Berry, Bellaire, TX 1941-06-20
```

SQL, Relational algebra, and Relational calculus

SELECT $A_1, A_2, \ldots, A_n$

FROM $R_1, R_2, \ldots, R_m$

WHERE P

- To Relational algebra:
  - $\pi_{A_1, A_2, \ldots, A_n}(\sigma_P(R_1 \times R_2 \times \ldots \times R_m))$
- To Relational calculus:
  - $\{t_1.A_1, t_2.A_2, \ldots, t_{m-k}.A_n \mid R_1(t_1) \wedge \ldots \wedge R_{m-k}(t_{m-k}) \wedge (\exists t_{m-k+1})(\exists t_m)(R_{m-k+1}(t_{m-k+1}) \wedge \ldots \wedge R_m(t_m) \wedge P)\}$

Query 0

SELECT BDATE, ADDRESS

FROM EMPLOYEE

WHERE FNAME='John' **AND** MINIT='B' **AND** LNAME='Smith';

- In Relational algebra
  - $\pi_{\text{BDATE, ADDRESS}}(\sigma_{\text{FNAME='John'} \wedge \text{MINIT='B'} \wedge \text{LNAME='Smith'}}(\text{EMPLOYEE}))$
- In Relational calculus
  - {t.BDATE, t.ADDRESS | EMPLOYEE(t) $\wedge$ FNAME='John' $\wedge$ MINIT='B' $\wedge$ LNAME='Smith'}

Query 1

SELECT FNAME, LNAME, ADDRESS

FROM EMPLOYEE, DEPARTMENT

WHERE DNAME='Research' **AND** DNUMBER=DNO

- In Relational algebra
  - $\pi_{FNAME,LNAME,ADDRESS}(\sigma_{DNAME='Research' \wedge DNUMBER=DNO}(EMPLOYEE \times DEPARTMENT))$
  - $\pi_{FNAME,LNAME,ADDRESS}(\sigma_{DNAME='Research'}(EMPLOYEE \bowtie_{DNUMBER=DNO} DEPARTMENT))$
- In Relational calculus
  - {t.FNAME, t.LNAME, t.ADDRESS |
    EMPLOYEE(t) $\wedge$ ($\exists$d)(DEPARTMENT(d) $\wedge$
    d.DNAME='Research' $\wedge$ d.DNUMBER=t.DNO)}

Queries without selection or projection

- Missing WHERE clause
  - No selection
  - All tuples of the table from FROM clause are selected
  - If more than 1 table, the result is Cross product
- Use of Asterisk (*)
  - No projection
  - Retrieves all attribute values of selected tuples

Query 3 (query without selection)

- Select all names of departments

```
SELECT   DNAME
FROM     DEPARTMENT;
```

- Result

```
DNAME
===============
Headquarters
Administration
Research
```

Query 4 (cross product)

- Select all combinations of employees' ssn and department names

SELECT SSN, DNAME
FROM EMPLOYEE, DEPARTMENT;

- Result

```
SSN       DNAME
========= ==============
123456789 Headquarters
333445555 Headquarters
453453453 Headquarters
666884444 Headquarters
888665555 Headquarters
987654321 Headquarters
987987987 Headquarters
999887777 Headquarters
123456789 Administration
333445555 Administration
453453453 Administration
666884444 Administration
888665555 Administration
987654321 Administration
987987987 Administration
999887777 Administration
123456789 Research
333445555 Research
453453453 Research
666884444 Research
888665555 Research
987654321 Research
987987987 Research
999887777 Research
```

Query 5 (using asterisk)

- Retrieve all attribute values for employee named ‘Narayan’

SELECT *
FROM EMPLOYEE
WHERE LNAME= ‘Narayan’

- Result

```
FNAME    MINIT LNAME    SSN         BDATE
ADDRESS                      SEX SALARY    SUPERSSN
  DNO
=============================================================
Ramesh   K     Narayan 666884444 1962-09-15
975 Fire Oak, Humble, TX  M   38000.00 333445555
    5
```

Prefixing, aliasing, renaming

- Prefix attribute name with table name
  - table_name.attribute_name
  - in **SELECT** and **WHERE** clauses
    - same attribute names from different relations in a query
- Introduce tuple variable for each relation
  - table_name **AS** new_name
  - in **FROM** clause
    - recursive query (join relation with itself)
- Rename attribute name
  - in **SELECT** clause
    - attribute_name **AS** new_name

Query 1A (Prefixing example)

- Suppose
  - LNAME of EMPLOYEE called NAME
  - DNAME of DEPARTMENT called NAME
- Retrieve the last name and address of all employees who work for the ‘Research’ department

SELECT EMPLOYEE.NAME, ADDRESS

FROM EMPLOYEE, DEPARTMENT

WHERE DEPARTMENT.NAME=‘Research’ **AND** DNUMBER=DNO;

Query 6 (Tuple variables)

- For each employee, retrieve the employee's first and last name and the first and last name of his/her immediate supervisor

SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME
FROM EMPLOYEE **AS** E, EMPLOYEE **AS** S
WHERE E.SUPERSSN=S.SSN;

- Result

FNAME	LNAME	FNAME	LNAME
John	Smith	Franklin	Wong
Franklin	Wong	James	Borg
Joyce	English	Franklin	Wong
Ramesh	Narayan	Franklin	Wong
Jennifer	Wallance	James	Borg
Ahmad	Jabbar	Jennifer	Wallance
Alicia	Zelaya	Jennifer	Wallance

Query 6 (renaming, SELECT clause)

SELECT E.FNAME **AS** E_FNAME, E.LNAME **AS** E_LNAME, S.FNAME **AS** S_FNAME, S.LNAME **AS** S_LNAME

FROM EMPLOYEE **AS** E, EMPLOYEE **AS** S

WHERE E.SUPERSSN=S.SSN;

- Result

```
E_FNAME      E_LNAME      S_FNAME      S_LNAME
===========  ===========  ===========  =========
```

…

Duplicate elimination in SQL

- SQL does not automatically eliminates duplicates
  - it is expensive
  - user wants to use duplicates
  - when aggregate function is applied duplicates are wanted
    - could be specified explicitly by **SELECT ALL** …
- To eliminate duplicates specify
  - **SELECT DISTINCT** …

Query 7: retrieve the location of every project

SELECT PLOCATION
FROM PROJECT;

- Result

```
PLOCATION
===========
Bellaire
Sugarland
Houston
Stafford
Houston
Stafford
```

SELECT DISTINCT PLOCATION
FROM PROJECT;

- Result

```
PLOCATION
============
Bellaire
Houston
Stafford
Sugarland
```

Set operation in SQL

- Set opreations
  - UNION – set union
  - EXCEPT – set difference
  - INTERSECT – set intersection
    - table1 OP table2
- Duplicates are eliminated
  - use ALL to keep duplicates
    - UNION ALL, EXCEPT ALL, INTERSECT ALL
- Applied only to union-compatible tables

Query 8 (set operations)

- Make a list of all project numbers for projects that involve an employee whose name is ‘Smith’, either as a worker or as a manager of the department that controls the project

```
(SELECT     DISTINCT PNUMBER
FROM        PROJECT, DEPARTMENT, EMPLOYEE
WHERE       DNUM=DNUMBER AND MGRSSN=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
```

Temporal data types

- DATE ‘yyyy-mm-dd’
- TIME ‘hh:mm:ss’
  - TIME WITH TIME ZONE ‘hh:mm:ss +hh:mm’
- TIMESTAMP ‘yyyy-mm-dd hh:mm:ss ffffff’
  - with time zone
  - e.g., TIMESTAMP ‘2002-09-27 09:12:47 648302’
- INTERVAL – a relative value
  - e.g., INTERVAL ‘1’ DAY

Operations

- Arithmetic operators:
  - addition (+), subtraction (-), multiplication (*), division (/)
- String operator
  - concatenation (||) of two strings
- Temporal
  - incrementing (+), decrementing (-) time, date, timestamp by interval data types
- Can be used in SELECT and WHERE clauses
  - use rename for result column with arithmetic operation

Query 9 (arithmetic operation)

- Show the resulting salaries if every employee working on the ‘ProductX’ project is given a 10 percent raise

```
SELECT    FNAME, LNAME,
          1.1*SALARY AS INC_SAL
FROM      EMPLOYEE, WORKS_ON, PROJECT
WHERE     SSN=ESSN AND PNO=PNUMBER AND
  PNAME=‘ProductX’;
```

- Result

```
FNAME           LNAME                INC_SAL
=============== =============== ==============
John            Smith                33000.000
Joyce           English              27500.000
```

Specialized comparison operators

- Matching strings with patters
  - use comparison operator **LIKE**
  - % for any number of arbitrary symbols
  - _ for any symbol
- Check that numerical value is inside an interval
  - Comparison operator **BETWEEN**
    - attribute **BETWEEN** value1 **AND** value2
    - (attribute >= value1) **AND** (attribute <= value2)

Query 10 (using LIKE)

- Retrieve all employees whose address is in Houston, Texas

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

- Result

```
FNAME           LNAME
=============== ===============
John            Smith
Franklin        Wong
James           Borg
Ahmad           Jabbar
```

Query 11 (using BETWEEN)

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

```
SELECT    LNAME, SALARY
FROM      EMPLOYEE
WHERE     (SALARY BETWEEN 30000 AND 40000)
  AND DNO=5;
```

- Result

```
LNAME                        SALARY
================ ============
Smith                     30000.00
Wong                      40000.00
Narayan                   38000.00
```

Ordering result

- The tuples in the result can be ordered by the values of one or more attributes
  - use **ORDER BY** clause
  - tuples are ordered by first attribute than they are ordered within same value of the attribute by second attribute, and so on
- Order can be specified by
  - **ASC** – ascending order (default)
  - **DESC** – descending order

Query 12 (using ORDER BY)

- Retrieve a list of employees in the ascending order of their first name

SELECT FNAME, LNAME

FROM EMPLOYEE

ORDER BY FNAME;

- Result

FNAME	LNAME
Ahmad	Jabbar
Alicia	Zelaya
Franklin	Wong
James	Borg
Jennifer	Wallance
John	Smith
Joyce	English
Ramesh	Narayan

Query 13 (using DESC)

- Retrieve all employees and their salary ordered by their salary in descendent order within each salary by they last name

```
SELECT  FNAME, LNAME,
                SALARY
FROM      EMPLOYEE
ORDER BY SALARY DESC,
                LNAME;
```

- Result

FNAME	LNAME	SALARY
James	Borg	55000.00
Jennifer	Wallance	43000.00
Franklin	Wong	40000.00
Ramesh	Narayan	38000.00
John	Smith	30000.00
Joyce	English	25000.00
Ahmad	Jabbar	25000.00
Alicia	Zelaya	25000.00

NULL Values

- Each NULL is unique (except grouping)
- Three-valued logic: TRUE, FALSE, UNKNOWN
- Result of queries contain only those row for which the condition is TRUE
- Check for NULL value
  - **IS** NULL
  - **IS NOT** NULL

Query 14 (using IS NULL)

- Retrieve the names of all employees who do not have supervisors

```
SELECT   FNAME, LNAME
FROM     EMPLOYEE
WHERE    SUPERSSN IS NULL;
```

- Result

```
FNAME                LNAME
=============== ===============
James                Borg
```

Three-valued logic: AND, OR, NOT

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

NOT	TRUE	FALSE	UNKNOWN
	FALSE	TRUE	UNKNOWN

Nested queries

- Complete select-from-where block (nested query) within **WHERE** clause of another query (outer query)
- Check if a tuple is contained by the result of nested query
  - attribute **IN** nested_query
  - = **ANY** and = **SOME**
- Comparison operators
  - >,<,>=,<=,=,<> with **ANY**, **SOME**, **ALL**
- Nested query is evaluated once for each tuple in the outer query

Query 15 (nested query)

- Retrieve SSN of all employees who work on the same project as employee with SSN=123456789

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

- Result

```
ESSN
=========
123456789
333445555
453453453
```

Query 16 (>ALL)

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

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

- Result

```
LNAME           FNAME
=============== ===============
Borg            James
Wallance        Jennifer
```

Nested queries

- Several levels of nested queries can be used
- Unqualified attribute refers to the relation declared in the innermost nested query
  - always create tuple variables to avoid potential errors and ambiguities
- Correlated nested queries
  - an attribute of outer query is referred in WHERE clause of nested query
- Queries written with nested query and using **IN** can be rewritten with single block query

Query 17 (correlated nested query)

- Retrieve the name of each employee who has a dependent with the same first name as the employee

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

- Rewritten query

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

EXISTS

- EXISTS checks if result of nested query is not empty
  - NOT EXISTS – opposite
- EXISTS are usually used in conjunction with correlated nested queries

Query 17B (query 17 with using EXISTS)

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

Query 18 (using NOT EXISTS)

- Retrieve the names of employee who have no dependents

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

Join

- Query with more than one table
  - has always join between them
  - join conditions specified to avoid cross product
- Explicit join in FROM clause
  - to specify different type of join
  - to specify join condition together with join
  - table1 (INNER) JOIN table2 ON join condition
    - default
  - LEFT/RIGHT/FULL (OUTER) JOIN
  - NATURAL (INNER) JOIN
    - no condition (join on attributes with that same name)
    - with LEFT/RIGHT/FULL (OUTER) JOIN

Query 6 (using OUTER JOIN)

- Retrieve employee's name with name of his supervisor

```
SELECT E.LNAME AS
    E_NAME, S.LNAME AS
    S_NAME
FROM (EMPLOYEE AS E
    LEFT OUTER JOIN
    EMPLOYEE AS S ON
    E.SUPERSSN=S.SSN);
```

- Result

E_NAME	S_NAME
Smith	Wong
Wong	Borg
English	Wong
Narayan	Wong
Borg	-
Wallance	Borg
Jabbar	Wallance
Zelaya	Wallance

Aggregate functions

- Functions
  - COUNT for rows
  - SUM, AVG numerical domain
  - MAX, MIN domains with total ordering
- NULL values discarded during applying aggregations on an attribute
- Used in SELECT and HAVING clauses

Query 19 (aggregate functions)

- Find the sum, max, min and avg of the salaries of all employees of the ‘Research’ department

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

- Result

```
===========  =========  ==========  =========
  133000.00   40000.00    25000.00   33250.00
```

Query 20 (using COUNT)

- Retrieve the total number of employees in the company

```
SELECT   COUNT(*)
FROM     EMPLOYEE;
```

- Result

```
============
```

```
8
```

Grouping

- All result tuples are split to subgroups based on grouping attributes
  - Tuples are in the same subgroup if values of grouping attributes are the same
  - Separate subgroup for tuples with values NULL
- Grouping attributes defined in **GROUP BY** clause
  - Aggregate functions should be applied to all non-grouping attributes in SELECT clause

Query 21 (grouping example)

- 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;
```

- Result

```
DNO
====== ======== =============
     1        1    55000.000
     4        3    31000.000
     5        4    33250.000
```

Condition on group selection

- Retrieve groups that satisfy certain condition
  - in **HAVING** clause
- **HAVING** clause is used in conjunction with **GROUP BY** clause only

Query 22 (using HAVING)

- For each project on which more than two employees work, retrieve the project number, its name, and the number of its employees

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

- Result

```
 PNUMBER PNAME
========= =============== ===========
        2 ProductY                  3
       10 Computerization           3
       20 Reorganization            3
       30 Newbenefits               3
```

Summary

- Clauses:

  SELECT <attribute list>

  FROM <table list>

  [**WHERE** <condition>]

  [**GROUP BY** <grouping attributes>

  [**HAVING** <group condition>]]

  [ORDER BY <attribute list>**]**

- Numerous ways to specify the same query