SQL

Database Systems (H)

Dr Chris Anagnostopoulos

ROADMAP

Ta Dah!

- **Structured Query Language (SQL);**
  - Create a database *schema* & *relations* in SQL;
  - Assign key/integrity/referential *constraints* in SQL;

- **SELECT clause for *selection* queries;**
  - *Multi-sets* and *Sets* in SQL
  - Dealing with NULL values

- **Nested Correlated & Uncorrelated Queries**

PHILOSOPHY OF THE DECLARATIVE LANGUAGE

- Structured Query Language *by* R. Boyce (*known* from BCNF) in **1974**.

- SQL is a **declarative** language, i.e.,
  - declare *what to do* rather than *how to do it*
  - different from procedural languages, e.g., Java, C, C++.

- First *official standard*: **SQL-92**
- Latest release: **SQL:2016**…
- **Advice:** *follow* standard SQL to be compliant with *most* of the Database Systems 🏝

SQL: DATABASE SCHEMA

- Statement: `CREATE SCHEMA`

  `CREATE SCHEMA` Company;

Each statement in SQL *ends* with a semicolon ‘;’

SQL: CREATE TABLE

- A *new* relation (*table* is SQL):
  - Specify the *name* of the relation
  - Specify *attributes*, their *types* (domain), *constraints*

```
CREATE SCHEMA Company;
CREATE TABLE EMPLOYEE ...;
```

SQL: ATTRIBUTES & DOMAINS

- **Numeric data types**
  - Integer numbers: **INT**
  - Floating-point (real) numbers: **REAL** or **DECIMAL(*n*, *m*)**
  - DECIMAL(3,2) has 3 digits; 2 digits after the decimal '.' e.g., 9.99

- **Character/String data types**
  - Fixed length: **CHAR(*n*)**
    - i.e., exactly *n* characters
    - e.g., CHAR(5) has exactly 5 characters like 'Chris'
  - *Variable* length: **VARCHAR(*n*)**
    - i.e., from 0 to *n* characters
    - e.g., VARCHAR(5) has up to 5 characters like 'C', or, 'Ch', or 'Chris'

SQL: ATTRIBUTES & DOMAINS

- **Bit-string data types (*sequence* of bits: e.g., 0101100)**
  - *Fixed* length: **BIT(*n*)**
  - *Varying* length: **BIT VARYING(*n*)**

- **Boolean data type**
  - Values of `TRUE` or `FALSE` or `NULL`
  - `SQL` is a 3-valued *logic*…(*yes*, *no*, and *maybe*)

- **`DATE` data type**
  - *Ten* positions for YEAR, MONTH, and DAY in the form `YYYY-MM-DD`

- **More,** like **TIMESTAMP, DATE INTERVALS, …**

  Visit: https://www.postgresql.org/docs/9.5/static/datatype.html

SQL: CREATE TABLE

```
CREATE TABLE EMPLOYEE
        ( Fname          VARCHAR(15)      NOT NULL,
          Minit          CHAR,
          Lname          VARCHAR(15)      NOT NULL,
          Ssn            CHAR(9)          NOT NULL,
          Bdate          DATE,
          Address        VARCHAR(30),
          Sex            CHAR,
          Salary         DECIMAL(10,2),
          Super_ssn      CHAR(9),
          Dno            INT              NOT NULL,
        PRIMARY KEY (Ssn),
CREATE TABLE DEPARTMENT
        ( Dname          VARCHAR(15)      NOT NULL,
          Dnumber        INT              NOT NULL,
          Mgr_ssn        CHAR(9)          NOT NULL,
          Mgr_start_date DATE,
        PRIMARY KEY (Dnumber),
        UNIQUE (Dname),
        FOREIGN KEY (Mgr_ssn) REFERENCES EMPLOYEE(Ssn) );
CREATE TABLE DEPT_LOCATIONS
        ( Dnumber        INT              NOT NULL,
          Dlocation      VARCHAR(15)      NOT NULL,
        PRIMARY KEY (Dnumber, Dlocation),
        FOREIGN KEY (Dnumber) REFERENCES DEPARTMENT(Dnumber) );
```

attributes

domain (type)

constraints

SQL: VALUE CONSTRAINTS

- *Default* value of an attribute
  - **DEFAULT** {value}
  - **NULL** is not permitted for a attribute (**NOT NULL**)
  - e.g., DNO **INT NOT NULL DEFAULT 1;**

- **CHECK** clause (*range domain* constraint)
  - e.g., Dnumber **INT NOT NULL CHECK**(Dnumber > 0 **AND** Dnumber < 21);

SQL: KEY CONSTRAINTS

- **Key constraint**: a **primary key value** is unique (no duplicates);
- **Entity Integrity constraint**: a primary key cannot be `NULL`;

- Primary Key Clause:
  - Dnumber **INT NOT NULL, PRIMARY KEY** (Dnumber);

- **UNIQUE** clause, specifies *candidate* keys
  - Dname **VARCHAR(15) NOT NULL, UNIQUE** (Dname);

```
CREATE TABLE DEPARTMENT
        ( Dname            VARCHAR(15)        NOT NULL,
          Dnumber          INT                NOT NULL,
          Mgr_ssn          CHAR(9)            NOT NULL,
          Mgr_start_date   DATE,
        PRIMARY KEY (Dnumber),
        UNIQUE (Dname),
        FOREIGN KEY (Mgr_ssn) REFERENCES EMPLOYEE(Ssn) );
```

SQL: REFERENTIAL CONSTRAINTS

- FOREIGN KEY clause in EMPLOYEE
  FOREIGN KEY (Super_ssn) REFERENCES Employee(Ssn)

- FOREIGN KEY clause in DEPARTMENT
  FOREIGN KEY (Mgr_ssn) REFERENCES Employee(Ssn)

- *Triggered* actions for **Mgr_ssn, Super_ssn** when **Ssn** is updated or deleted:
  - Action: **ON DELETE SET NULL/ DEFAULT/ CASCADE**
  - Action: **ON UPDATE SET NULL/ DEFAULT /CASCADE**

- CASCADE option *propagates* DELETE / UPDATE to *all* referential tuples!

  - *e.g.*, when the primary key **Ssn** is updated, then all foreign keys *refer* to it should be updated: **ON UPDATE CASCADE**

  - *e.g.*, when the primary key **Ssn** is deleted, then all foreign keys *refer* to this tuple should be deleted: **ON DELETE CASCADE**

IN-CLASS QUIZ

Q1: What is happening when we *delete* a department?

Q2: What is happening when we *delete* an employee?

```
CREATE TABLE EMPLOYEE
    ( … ,
      Dno          INT          NOT NULL      DEFAULT 1,
    CONSTRAINT EMPPK
      PRIMARY KEY (Ssn),
    CONSTRAINT EMPSUPERFK
      FOREIGN KEY (Super_ssn) REFERENCES EMPLOYEE(Ssn)
                    ON DELETE SET NULL          ON UPDATE CASCADE,
    CONSTRAINT EMPDEPTFK
      FOREIGN KEY(Dno) REFERENCES DEPARTMENT(Dnumber)
                    ON DELETE SET DEFAULT       ON UPDATE CASCADE);
CREATE TABLE DEPARTMENT
    ( … ,
      Mgr_ssn CHAR(9)          NOT NULL      DEFAULT '888665555',
      … ,
    CONSTRAINT DEPTPK
      PRIMARY KEY(Dnumber),
    CONSTRAINT DEPTSK
      UNIQUE (Dname),
    CONSTRAINT DEPTMGRFK
      FOREIGN KEY (Mgr_ssn) REFERENCES EMPLOYEE(Ssn)
                    ON DELETE SET DEFAULT       ON UPDATE CASCADE);
CREATE TABLE DEPT_LOCATIONS
    ( … ,
    PRIMARY KEY (Dnumber, Dlocation),
    FOREIGN KEY (Dnumber) REFERENCES DEPARTMENT(Dnumber)
                    ON DELETE CASCADE           ON UPDATE CASCADE);
```

SELECT-FROM-WHERE

```
SELECT    <attribute list>
FROM      <table list>
WHERE     <condition>;
```

- Declare *what* to retrieve, i.e., which are the **attributes of interest**
- Declare *from* where to retrieve, i.e., which is the **table/relation**
- Declare with what *condition* to retrieve, i.e., which are the **conditions**

But, not saying how to *implement* this, e.g.,

- *how* to load the data from disk to memory,
- *how* to *search* and *check* if a tuple satisfies the condition, etc.

SELECT-FROM-WHERE

Query 0: Which are the addresses of employees working in the department 4 *or* their salary is less that 31000.

SELECT Address
FROM EMPLOYEE
WHERE DNO = 4 **OR** Salary < 31000

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	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

SELECT-FROM-WHERE: JOIN & SELECT

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

```
Q1:   SELECT   Fname, Lname, Address
      FROM     EMPLOYEE, DEPARTMENT
      WHERE    Dname='Research' AND Dnumber=Dno;
```

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	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

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

Fname	Lname	Address
John	Smith	731 Fondren, Houston, TX
Franklin	Wong	638 Voss, Houston, TX
Ramesh	Narayan	975 Fire Oak, Humble, TX
Joyce	English	5631 Rice, Houston, TX

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 birth date.

```
Q2:   SELECT   Pnumber, Dnum, Lname, Address, Bdate
      FROM     PROJECT, DEPARTMENT, EMPLOYEE
      WHERE    Dnum=Dnumber AND Mgr_ssn=Ssn AND
               Plocation='Stafford';
```

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	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

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

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

Pnumber	Dnum	Lname	Address	Bdate
10	4	Wallace	291Berry, Bellaire, TX	1941-06-20
30	4	Wallace	291Berry, Bellaire, TX	1941-06-20

TABLE AS A VARIABLE

//in Java.

```
int e = 5;
int s = 7;
```

//in SQL

```
EMPLOYEE AS e
EMPLOYEE AS s
```

- A relation might play different *roles* within a query, e.g., employee *might* be a *supervisee* and employee might be a *supervisor*…(used in recursive references…)

SELECT …

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

WHERE…

Query 3. 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;
```

EMPLOYEE **E**

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	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

{John Smith, Franklin Wong}

EMPLOYEE **S**

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	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

IN-CLASS ACTIVITY [A1]

Query 4: For each *employee*, if their *supervisor* is a *manager* of a *department*, show the department name and department number.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	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

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

IN-CLASS ACTIVITY [A1]

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	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

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

- **John** is supervised by **33344555**, who is **Franklin**. Franklin is manager of department **Research** (No. **5**).
- **Franklin** is supervised by **888665555**, who is **James**. James is manager of **HQ** (No. **1**)
- …

```
SELECT  D.Dname, D.Dnumber
FROM    EMPLOYEE AS E, EMPLOYEE AS S, DEPARTMENT AS D
WHERE   E.Super_ssn=S.Ssn AND S.Ssn = D.Mgr_Ssn;
```

IF WHERE IS MISSING…

R. Cartesius;1648

- Missing WHERE: *no condition* on tuple selection
- If FROM involves *two* or *more* relations, **avoid**; *unreasonable* tuples.
- **Why?** CROSS (*Cartesian*) PRODUCT: *all* possible tuple combinations!

```
SELECT    Ssn
FROM      EMPLOYEE;

SELECT    Ssn, Dname
FROM      EMPLOYEE, DEPARTMENT;
```

Each *tuple* from **EMLOYEE** is *concatenated* with *each* tuple from **DEPARTMENT**…disaster, computationally heavy, and meaningless!

MISSING WHERE IS CATASTROPHE

R. Cartesius;1648

SELECT Ssn, Dname

FROM EMPLOYEE, DEPARTMENT

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	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

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

John…Research
John…Administration
John…HQ
Franklin…Research
Fraklin…Administration
…

USE OF THE ASTERISK

If *bored* listing *all* the attributes, then *use* asterisk (*), i.e., *all* attributes are of interest

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

Select all the information about **those** employees working at the department 5

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

Select all the information (employee and department) from **those** employees working at the department 'Research'

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

Select all the information about employees and departments **with no meaning**

TABLES AS MULTI-SETS IN SQL

- **Set**: has only unique elements, e.g., S = {a, b, c}
- **Multiset**: might have duplicates, e.g., M = {a, a, a, b, c, c}
- Operators: `UNION, EXCEPT, INTERSECT`

Query 5: Retrieve the salary of *each* employee, and *all* the distinct salaries

```
SELECT    Salary
FROM      EMPLOYEE;

SELECT    DISTINCT Salary
FROM      EMPLOYEE;
```

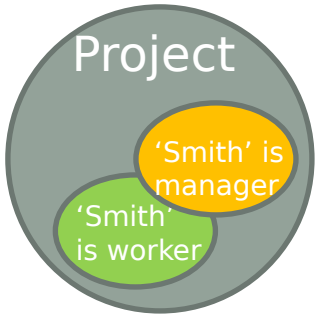
Salary
10000
10000
25000
30000
25000
30000
30000

Salary
10000
25000
30000

IN-CLASS ACTIVITY [A2]

List *all* project numbers for projects that involve employees, whose last name is ‘Smith’, *either* as **workers** *or* as **managers** of departments controlling these projects.

Idea: *split* this into *two* sub-queries and then use the *set* UNION operator over the partial results.



IN-CLASS ACTIVITY [A2]

Step 1: Retrieve the projects where an employee with surname ‘Smith’ is *working* on;

Associate **EMLOYEE** with **PROJECT** via **WORKS_ON**

```
( SELECT    DISTINCT Pnumber
  FROM      PROJECT, WORKS_ON, EMPLOYEE
  WHERE     Pnumber=Pno AND Essn=Ssn
            AND Lname='Smith' );
```

IN-CLASS ACTIVITY [A2]

Step 2: Retrieve the projects where an employee with surname ‘Smith’ is a *manager* of the department which controls these project(s);

Associate **EMLOYEE** with **DEPARTMENT** to get the manager, and *then* **DEPARTMENT** with **PROJECT** to get the controlled projects by this department.

```
(SELECT   DISTINCT Pnumber
 FROM     PROJECT, DEPARTMENT, EMPLOYEE
 WHERE    Dnum=Dnumber AND Mgr_ssn=Ssn
          AND Lname='Smith' )
```

IN-CLASS ACTIVITY [A2]

Step 3: `UNION` over the two *sets* of project numbers:

```
( 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' );
```

THREE-VALUED LOGIC

SQL is a three-valued logic: `TRUE (1)`, `FALSE (0)` and `UNKNOWN(0.5)`

Recall: Each NULL value is *different* from any other NULL value!

Principle: *Any* value compared with `NULL` evaluates to `UNKNOWN`

Example:

WHERE Address = `NULL` ...evaluates to `UNKNOWN;`

WHERE Address <> `NULL` ...evaluates to `UNKNOWN;`

WHERE `NULL` = `NULL` ...evaluates to `UNKNOWN`

Always adopt: **IS NULL** or **IS NOT NULL**

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

min

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

max

NOT	
TRUE	FALSE
FALSE	TRUE
UNKNOWN	UNKNOWN

1-*x*

COMPARISON INVOLVING NULL

Query 6: Retrieve the first and last names of *all* employees who do not have supervisors.

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

```
SELECT  Fname, Lname
FROM    EMPLOYEE
WHERE   Super_ssn = NULL
```

it produces *no* tuples!
Hence, *wrong* reasoning!
Why?

IN-CLASS ACTIVITY [A3]

```
CREATE TABLE Recruitment(
 ID INT NOT NULL,
 Name VARCHAR(10),
 RecruiterID INT,
 PRIMARY KEY(ID),
 FOREIGN KEY(RecruiterID)
 REFERENCES Recruitment(ID)
);
```

ID	Name	RecruiterID
1	Chris	NULL
2	Philip	3
3	Stella	NULL
4	John	2
5	Teresa	3
6	Iona	5

Task 1: List *all* the names of the employees.

Task 2: List the names of the employees who are *not* recruited by employee with ID = 2.

Which is the problem here?

IN-CLASS ACTIVITY [A3]

{Chris, Philip, Stella, John, Teresa, Iona}

```
SELECT Name, RecruiterID
FROM   Recruitment
WHERE  RecruiterID  <> 2
```

{Philip, Teresa, Iona}

We *imply*…{Chris, Stella, John} are recruited by Philip.

```
SELECT Name, RecruiterID
FROM   Recruitment
WHERE  RecruiterID IS NULL
    OR RecruiterID  <> 2
```

{Chris, Philip, Stella, Teresa, Iona}

ID	Name	RecruiterID
1	Chris	NULL
2	Philip	3
3	Stella	NULL
4	John	2
5	Teresa	3
6	Iona	5

NESTED (INNER) QUERY

Nested query is a query *within* another (*outer*) query;

- SELECT-FROM-WHERE block *within* another outer WHERE clause.
- Nested query's *output* is *input* to outer's WHERE via: **IN**, **ALL**, **EXISTS**
- **Nested Uncorrelated Query**: *first* execute the nested query, and *then* execute the outer query using inner's output.
- **Correlated Query**: for *each* tuple of the outer query, we execute the nested query.

NESTED UNCORRELATED QUERY: OPERATOR **IN**

IN: checks whether a value belongs to the inner's output *set* (or *multiset*), i.e.,

Query 7: Show the SSN of those employees who work in the projects with number: either 1, or 2, or 3.

```
SELECT  Essn
FROM    WORKS_ON
WHERE   PNO IN (1, 2, 3);
```

- if PNO = 1 then PNO **IN** (1, 2, 3) evaluates to `TRUE`
- if PNO = 4 then PNO **IN** (1, 2, 3) evaluates to `FALSE`

NESTED UNCORRELATED QUERY: OPERATOR **IN**

Query 8: Show the names of those employees who work in the department ‘Research’.

```
SELECT  FNAME
FROM    EMPLOYEE
WHERE   DNO IN ( SELECT  DNUMBER
                 FROM    DEPARTMENT
                 WHERE   DNAME = ‘Research’);
```

Evaluates to **5**

```
SELECT  FNAME
FROM    EMPLOYEE
WHERE   DNO IN ( 5 );
```

NESTED UNCORRELATED QUERY: OPERATOR **ALL**

ALL: compares a value with *all* the values from the inner's output *set.*

Query 9: Show the last and first names of those employees whose salary is *greater* than the salaries of *all* employees in Department 5.

```
SELECT   Lname, Fname
FROM     EMPLOYEE
WHERE    Salary > ALL  ( SELECT   Salary
                         FROM    EMPLOYEE
                         WHERE   Dno=5 );
```

First, find *all* salaries from employees in Department 5;

NESTED CORRELATED QUERY

For **each** *tuple* of the *outer* query, we execute the inner query!

Relation as a variable: **global scope** (*outer*) and **local scope** (*inner*).

Query 10: Retrieve the name of *each* employee who has a dependent with the same first name and is the same gender as that 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 );
```

For each *outer* employee E, retrieve the dependents D and check!

NESTED CORRELATED QUERY

Lemma 1: Correlated queries using **IN** can be collapsed into one *single* block.

Query 11: Retrieve the name of *each* employee who has a dependent with the same first name and is the same gender as that employee.

```
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;
```

NESTED CORRELATED QUERY: EXISTS

EXISTS: checks *whether* the inner's output is *empty* set or *not*, and returns FALSE or TRUE, respectively, e.g., or

Opposite: NOT EXISTS

```
SELECT E.Fname, E.Lname
FROM   EMPLOYEE AS E
WHERE  EXISTS
(SELECT * FROM DEPARTMENT AS D WHERE  E.DNO = D.DNUMBER)
```

- Checks if a *given* employee is working at *some* department.
- Reason about `E.DNO` being `NULL`.

NESTED CORRELATED QUERY: EXISTS

```
SELECT E.Fname, E.Lname
FROM   EMPLOYEE AS E
WHERE  EXISTS
       (SELECT *
        FROM   DEPARTMENT AS D
        WHERE  E.DNO = D.DNUMBER AND D.DNAME = 'Research')
```

- Describe this…

IN-CLASS QUIZ

```
SELECT E.Fname, E.Lname
FROM   EMPLOYEE AS E
WHERE EXISTS
(SELECT * FROM DEPENDENT AS P WHERE E.SSN = P.Essn)
 AND EXISTS
(SELECT  * FROM DEPARTMENT AS D WHERE E.SSN = D.Mgr_SSN)
```

Checks if a *given* employee:

- has at least a dependent **and**
- manages a department, i.e., there *exists* a department, which is managed by that employee.

IN-CLASS ACTIVITY [A4]

STUDENT (Name, <u>StudentID</u>, Class)

COURSE (Name, <u>CourseID</u>, Credits, School)

GRADES (<u>StudentID, CourseID</u>, Grade)

/*Grade: {‘A’, ‘B’, ‘C’, ‘D’, ‘E’}*/

Task: Retrieve the names of *all* students who have a Grade of ‘A’ in *all* of their courses.

IN-CLASS ACTIVITY [A4]

STUDENT (Name, StudentID, Class)

COURSE (Name, CourseID, Credits, School)

GRADES (StudentID, CourseID, Grade)

/*Grade: {'A', 'B', 'C', 'D', 'E'}*/

```
SELECT S.Name
FROM   STUDENT S
WHERE  NOT EXISTS
    (SELECT * FROM GRADES G
        WHERE G.StudentID = S.StudentID
              AND NOT (G.Grade = 'A')
        )
```

There does *not* exist a grade which is *not* 'A', i.e., *all* grades are 'A'

IN-CLASS QUIZ [WHAT WE GET]

```
SELECT *
FROM EMPLOYEE
WHERE  EXISTS (SELECT NULL FROM EMPLOYEE);

SELECT *
FROM EMPLOYEE
WHERE EXISTS (SELECT * FROM EMPLOYEE
        WHERE NULL IS NULL);
```

Fact: SQL, in evaluating EXISTS, *simply counts* rows and *ignores* the value(s) in the subquery—even if they are NULL!

IN-CLASS QUIZ [WHAT WE GET]

```
SELECT * FROM EMPLOYEE WHERE
      EXISTS (SELECT 0
           FROM EMPLOYEE
                  WHERE 1 IS NULL OR NULL);
```

```
SELECT * FROM EMPLOYEE WHERE
      NOT EXISTS (SELECT 0
                FROM EMPLOYEE
                       WHERE 1 IS NOT NULL OR NULL);
```