BASIC SQL

BASIC SQL

- **SQL language**
  - Considered one of the major reasons for the commercial success of relational databases
- **SQL**
  - SQL Actually comes from the word “SEQUEL” which was the original term used in the paper: “SEQUEL TO SQUARE” by Chamberlin and Boyce. IBM could not copyright that term, so they abbreviated to SQL and copyrighted the term SQL.
  - Now popularly known as “Structured Query language”.
  - SQL is an informal or practical rendering of the relational data model with syntax

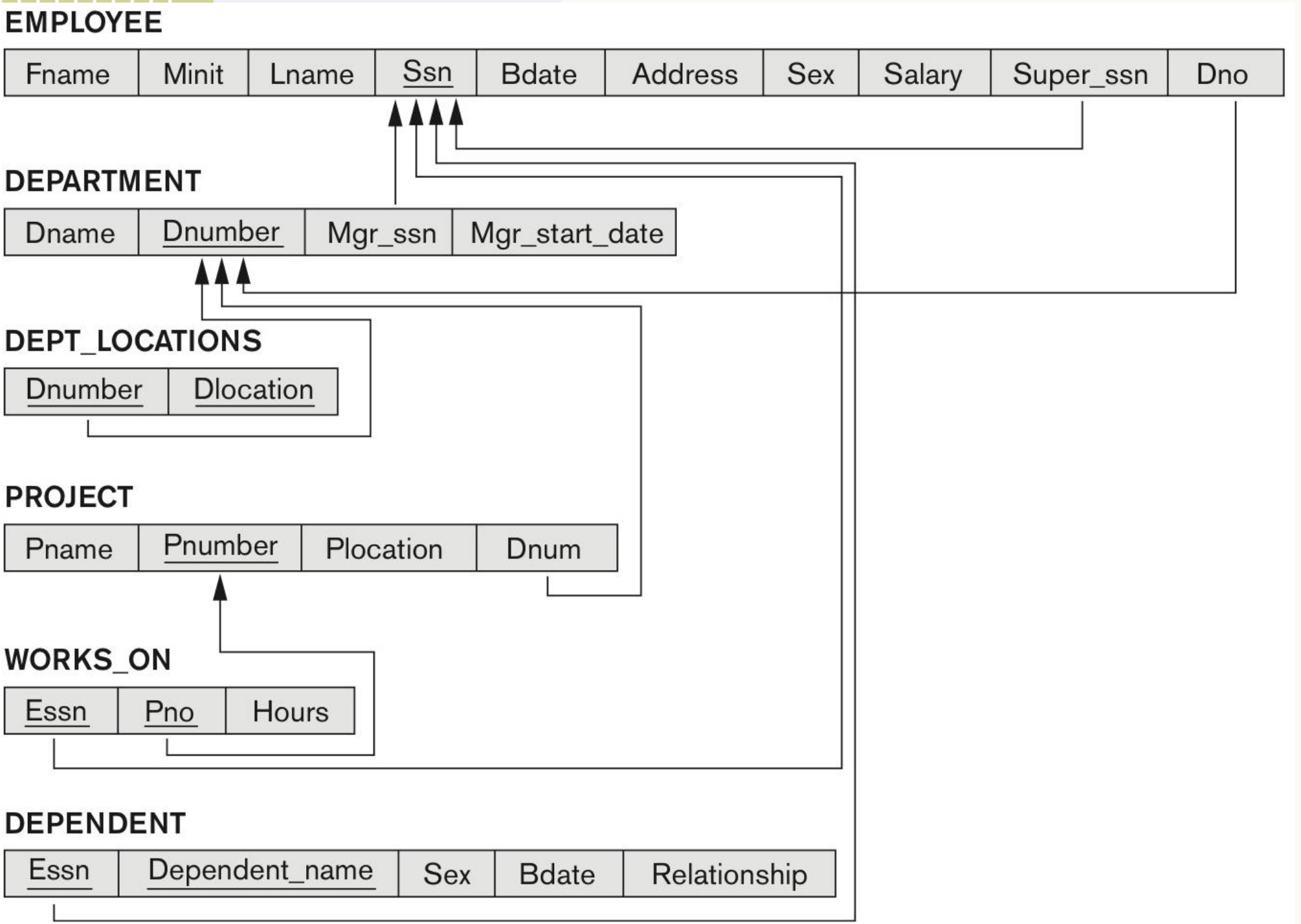
THE CREATE TABLE COMMAND IN SQL

- Specifying a new relation
  - Provide name of table
  - Specify attributes, their types and initial constraints
- Can optionally specify schema:

```
CREATE TABLE COMPANY.EMPLOYEE ...

  or

CREATE TABLE EMPLOYEE ...
```

THE CREATE TABLE COMMAND IN SQL (CONT'D.)

- **Base tables (base relations)**
  - Relation and its tuples are actually created and stored as a file by the DBMS
- **Virtual relations (views)**
  - Created through the `CREATE VIEW` statement. Do not correspond to any physical file.


EMPLOYEE
Fname
Minit
Lname
Ssn
Bdate
Address
Sex
Salary
Super_ssn
Dno
DEPARTMENT
Dname
Dnumber
Mgr_ssn
Mgr_start_date
DEPT_LOCATIONS
Dnumber
Dlocation
PROJECT
Pname
Pnumber
Plocation
Dnum
WORKS_ON
Essn
Pno
Hours
DEPENDENT
Essn
Dependent_name
Sex
Bdate
Relationship

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

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

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

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

```
CREATE TABLE PROJECT
        ( Pname                 VARCHAR(15)     NOT NULL,
          Pnumber               INT             NOT NULL,
          Plocation             VARCHAR(15),
          Dnum                  INT             NOT NULL,
        PRIMARY KEY (Pnumber),
        UNIQUE (Pname),
        FOREIGN KEY (Dnum) REFERENCES DEPARTMENT(Dnumber) );
CREATE TABLE WORKS_ON
        ( Essn                  CHAR(9)         NOT NULL,
          Pno                   INT             NOT NULL,
          Hours                 DECIMAL(3,1)    NOT NULL,
        PRIMARY KEY (Essn, Pno),
        FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn),
        FOREIGN KEY (Pno) REFERENCES PROJECT(Pnumber) );
CREATE TABLE DEPENDENT
        ( Essn                  CHAR(9)         NOT NULL,
          Dependent_name        VARCHAR(15)     NOT NULL,
          Sex                   CHAR,
          Bdate                 DATE,
          Relationship          VARCHAR(8),
        PRIMARY KEY (Essn, Dependent_name),
        FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn) );
```

CREATE TABLE EMPLOYEE (Fname VARCHAR(15) NOT NULL , Minit VARCHAR(15) NOT NULL, Lname VARCHAR(15) NOT NULL, Ssn CHAR(9) NOT NULL, Bdate DATE NOT NULL, Address VARCHAR(30), Sex CHAR NOT NULL, Salary DECIMAL(10,2) NOT NULL, Super_ssn CHAR(9) NOT NULL, Dno INT NOT NULL, PRIMARY KEY (Ssn));

SPECIFYING CONSTRAINTS IN SQL

Basic constraints:

- Relational Model has 3 basic constraint types that are supported in SQL:
  - **Key** constraint: A primary key value cannot be duplicated
  - **Entity Integrity** Constraint: A primary key value cannot be null
  - **Referential integrity** constraints : The “foreign key “ must have a value that is already present as a primary key, or may be null.

SPECIFYING ATTRIBUTE CONSTRAINTS

Other Restrictions on attribute domains:

- Default value of an attribute

```
DEFAULT <value>
```

- NULL is not permitted for a particular attribute (NOT NULL)
- **CHECK** clause

```
Dnumber INT NOT NULL CHECK (Dnumber > 0 AND
Dnumber < 21);
```

SPECIFYING KEY AND REFERENTIAL INTEGRITY CONSTRAINTS

- **PRIMARY KEY** clause
  - Specifies one or more attributes that make up the primary key of a relation

```
Dnumber INT PRIMARY KEY;
```

- **UNIQUE** clause
  - Specifies alternate (secondary) keys (called CANDIDATE keys in the relational model).
  -

```
Dname VARCHAR(15) UNIQUE;
```

SPECIFYING KEY AND REFERENTIAL INTEGRITY CONSTRAINTS (CONT'D.)

- **`FOREIGN KEY`** clause
  - Default operation: reject update on violation
  - Attach **referential triggered action** clause
    - Options include `SET NULL`, `CASCADE`, and `SET DEFAULT`
    - Action taken by the DBMS for `SET NULL` or `SET DEFAULT` is the same for both `ON DELETE` and `ON UPDATE`
    - `CASCADE` option suitable for "relationship" relations

GIVING NAMES TO CONSTRAINTS

- Using the Keyword **`CONSTRAINT`**
  - Name a constraint
  - Useful for later altering

```
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);
```

SPECIFYING CONSTRAINTS ON TUPLES USING CHECK

- Additional Constraints on individual tuples within a relation are also possible using CHECK
- `CHECK` clauses at the end of a `CREATE TABLE` statement
  - Apply to each tuple individually

```
CHECK (Dept_create_date <=
Mgr_start_date);
```

THE SELECT-FROM-WHERE STRUCTURE OF BASIC SQL QUERIES

- Basic form of the `SELECT` statement:

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

where

- <attribute list> is a list of attribute names whose values are to be retrieved by the query.
- <table list> is a list of the relation names required to process the query.
- <condition> is a conditional (Boolean) expression that identifies the tuples to be retrieved by the query.

THE SELECT-FROM-WHERE STRUCTURE OF BASIC SQL QUERIES (CONT'D.)

- Logical comparison operators
  - =, <, <=, >, >=, and <>
- **Projection attributes**
  - Attributes whose values are to be retrieved
- **Selection condition**
  - Boolean condition that must be true for any retrieved tuple. Selection conditions include join conditions when multiple relations are involved.

BASIC RETRIEVAL QUERIES

Bdate	Address
1965-01-09	731Fondren, Houston, TX

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 0. Retrieve the birth date and address of the employee(s) whose name is 'John B. Smith'.

```
Q0:   SELECT   Bdate, Address
      FROM     EMPLOYEE
      WHERE    Fname='John' AND Minit='B' AND Lname='Smith';
```

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;
```

BASIC RETRIEVAL QUERIES (CONTD.)

(c)

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

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';
```

AMBIGUOUS ATTRIBUTE NAMES

- Same name can be used for two (or more) attributes in different relations
  - As long as the attributes are in different relations
  - Must **qualify** the attribute name with the relation name to prevent ambiguity

```
Q1A:    SELECT    Fname, EMPLOYEE.Name, Address
        FROM      EMPLOYEE, DEPARTMENT
        WHERE     DEPARTMENT.Name='Research' AND
                  DEPARTMENT.Dnumber=EMPLOYEE.Dnumber;
```

ALIASING, AND RENAMING

- **Aliases** or **tuple variables**
  - Declare alternative relation names E and S to refer to the EMPLOYEE relation twice in a query:

Query 8. 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 E, EMPLOYEE  S WHERE E.Super_ssn=S.Ssn;
```

UNSPECIFIED WHERE CLAUSE AND USE OF THE ASTERISK

- Missing `WHERE` clause
  - Indicates no condition on tuple selection
- Effect is a `CROSS PRODUCT`
  - Result is all possible tuple combinations result

Queries 9 and 10. Select all EMPLOYEE Ssns (Q9) and all combinations of EMPLOYEE Ssn and DEPARTMENT Dname (Q10) in the database.

```
Q9:     SELECT    Ssn
        FROM      EMPLOYEE;

Q10:    SELECT    Ssn, Dname
        FROM      EMPLOYEE, DEPARTMENT;
```

UNSPECIFIED WHERE CLAUSE AND USE OF THE ASTERISK (CONT'D.)

- Specify an asterisk (*)
  - Retrieve all the attribute values of the selected tuples
  - The * can be prefixed by the relation name; e.g., EMPLOYEE *

```
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 does not automatically eliminate duplicate tuples in query results
- Use the keyword **`DISTINCT`** in the `SELECT` clause
  - Only distinct tuples should remain in the result

Query 11. Retrieve the salary of every employee (Q11) and all distinct salary values (Q11A).

```
Q11:    SELECT    ALL Salary
        FROM      EMPLOYEE;

Q11A:   SELECT    DISTINCT Salary
        FROM      EMPLOYEE;
```

TABLES AS SETS IN SQL (CONT'D.)

- Set operations
  - **UNION**, **EXCEPT** (difference), **INTERSECT**
  - Corresponding multiset operations: UNION ALL, EXCEPT ALL, INTERSECT ALL)
  - Type compatibility is needed for these operations to be valid

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

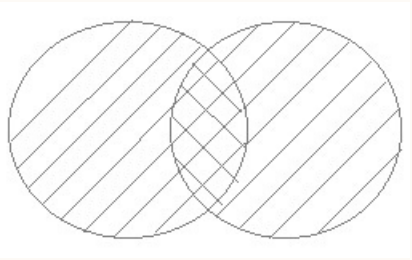
```
Q4A:  (SELECT    DISTINCT Pnumber
       FROM      PROJECT, DEPARTMENT, EMPLOYEE
       WHERE     Dnum=Dnumber AND Mgr_ssn=Ssn
                 AND Lname='Smith' )
       UNION
      ( SELECT   DISTINCT Pnumber
       FROM      PROJECT, WORKS_ON, EMPLOYEE
       WHERE     Pnumber=Pno AND Essn=Ssn
                 AND Lname='Smith' );
```

SET OPERATORS

- SQL set operators allows combine results from two or more SELECT statements.
- They are
- UNION
- UNION ALL
- INTERSECT
- MINUS (EXCEPT)

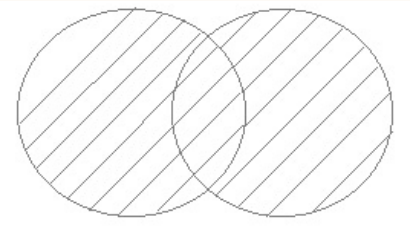
SET OPERATORS
UNION CLAUSE

- UNION clause/operator is used to combine the result-set of two or more SELECT statements
- SELECT statement within the UNION must have the same number of columns. With same data type
- UNION operator selects only distinct values by default.
- syntax
- SELECT *column_name(s)* FROM *table1*
  UNION
  SELECT *column_name(s)* FROM *table2*;



SET OPERATORS
UNION CLAUSE

- To allow duplicate values, use the ALL keyword with UNION.
- Syntax
- SELECT *column_name(s)* FROM *table1*
  UNION ALL
  SELECT *column_name(s)* FROM *table2*;



SET OPERATORS
UNION CLAUSE

Table test1

NAME	AGE	SEX
AJI	33	M
SAJI	36	M
RAJI	55	M
AJI	55	F

Table test2

NAME	AGE
AJI	25
SAJI	35
RAJI	30
AAJI	30
JI	33

SELECT NAME FROM TEST1 UNION SELECT NAME FROM TEST2;

NAME
AAJI
AJI
JI
RAJI
SAJI

5 rows returnec

SET OPERATORS
UNION ALL CLAUSE

Table test1

NAME	AGE
AJI	25
SAJI	35
RAJI	30
AAJI	30
JI	33

Table test2

NAME	AGE	SEX
AJI	33	M
SAJI	36	M
RAJI	55	M
AJI	55	F

SELECT NAME FROM TEST1 UNION ALL SELECT NAME FROM TEST2;

NAME
AJI
SAJI
RAJI
AAJI
JI
AJI
SAJI
RAJI
AJI

9 rows returne

UNION

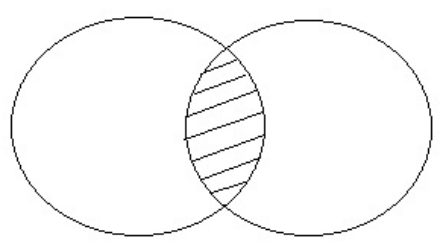
Write a SQL query to find all Location IDs from the Locations table that are located at any location, but also ensure that you include department IDs specifically located at the location with LOCATION_ID = '1100'. The query should remove any duplicates.

```
select L.LOCATION_ID from departments D, Locations
L where D.LOCATION_ID=L.Location_ID
union
select D.Department_ID from departments D,
Locations L where D.LOCATION_ID='1100';
```

	LOCATION_ID
1	120
2	1100
3	1400
4	1500
5	1700
6	1800
7	2400
8	2500
9	2700

SET OPERATORS
INTERSECT

- Returns any distinct values that are returned by both the query on the left and right sides of the INTERSECT operand.
- Syntax
  - SELECT *column_name(s)* FROM *table1*
    INTERSECT
    SELECT *column_name(s)* FROM *table2*;



SET OPERATORS
INTERSECT

Table test1

NAME	AGE
AJI	25
SAJI	35
RAJI	30
AAJI	30
JI	33

Table test2

NAME_S	SALARY
AJI	50
SAJI	40
AJI	10
SREE	10

```
SELECT NAME FROM TEST1 INTERSECT SELECT NAME_S
FROM TEST2;
```

NAME
AJI
SAJI

INTERSECT:

Finds jobs that are **currently assigned to employees.**

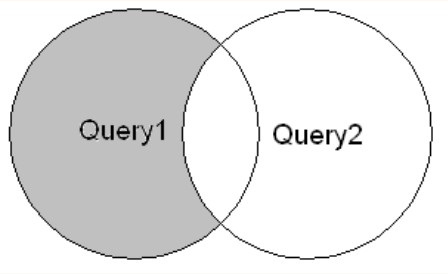
```
SELECT EMPLOYEES.JOB_ID
FROM EMPLOYEES

INTERSECT

SELECT JOBS.JOB_ID
FROM JOBS;
```

JOB_ID
AC_ACCOUNT
AC_MGR
AD_ASST
AD_PRES
AD_VP
FI_ACCOUNT
FI_MGR
HR_REP
MK_MAN
MK_REP
PR_REP
PU_CLERK
PU_MAN
SA_MAN
SA_REP
SH_CLERK
ST_CLERK
ST_MAN

SET OPERATORS
MINUS

- Returns any distinct values from the left query that are not also found on the right query.

```
select * from test1 where name='AJI’
        MINUS
select * from test2 where name_s='AJI'
```



SET OPERATORS
MINUS

Table test1

NAME	AGE
AJI	25
SAJI	35
RAJI	30
AAJI	30
JI	33

Table test2

NAME_S	SALARY
AJI	50
SAJI	40
AJI	10
SREE	10

SELECT NAME FROM TEST1 MINUS SELECT NAME_S FROM TEST2;

NAME
AAJI
JI
RAJI

3 rows retu

MINUS/EXCEPT

Write a query that find **departments that exist in the organization but currently have no employees.**

```
SELECT DEPARTMENT_ID
FROM Departments
MINUS
SELECT DEPARTMENT_ID
FROM Employees;
```

DEPARTMENT_ID
120
130
140
150
160
170
180
190
200
210
220
230
240
250
260
270

EXERCISES

1. Find all employee names and department names.
2. Find employees who are also listed as managers, assuming managers are listed in both employees and departments tables.
3. Find employees who are not listed in any department.
4. Identify employees who are not listed as managers in the departments table.

SUBSTRING PATTERN MATCHING AND ARITHMETIC OPERATORS

- **LIKE** comparison operator
  - Used for string **pattern matching**
  - % replaces an arbitrary number of zero or more characters
  - underscore (_) replaces a single character
- **BETWEEN** comparison operator

WHERE Address **LIKE** '%Houston,TX%'; **WHERE** Ssn **LIKE** '_ _ 1_ _ 8901';
WHERE(Salary **BETWEEN** 30000 **AND** 40000) **AND** Dno = 5;

ARITHMETIC OPERATIONS

- Standard arithmetic operators:
  - Addition (+), subtraction (–), multiplication (*), and division (/) may be included as a part of **SELECT**

Show the resulting salaries if every employee working on the 'ProductX' project is given a 10 percent raise.

```
SELECT  E.Fname, E.Lname, 0.1 * E.Salary AS Increased_sal
FROM  EMPLOYEE  E, WORKS_ON  W, PROJECT  P
WHERE  E.Ssn=W.Essn AND W.Pno=P.Pnumber AND P.Pname='ProductX';
```

ORDERING OF QUERY RESULTS

- Use **ORDER BY** clause
  - Keyword **DESC** to see result in a descending order of values
  - Keyword **ASC** to specify ascending order explicitly
  - Typically placed at the end of the query

```
ORDER BY D.Dname DESC, E.Lname ASC, E.Fname ASC
```

BASIC SQL RETRIEVAL QUERY BLOCK

```
SELECT      <attribute list>
FROM        <table list>
[ WHERE     <condition> ]
[ ORDER BY <attribute list> ];
```

INSERT, DELETE, AND UPDATE STATEMENTS IN SQL

- Three commands used to modify the database:
  - `INSERT`, `DELETE`, and `UPDATE`
- `INSERT` typically inserts a tuple (row) in a relation (table)
- `UPDATE` may update a number of tuples (rows) in a relation (table) that satisfy the condition
- `DELETE` may also update a number of tuples (rows) in a relation (table) that satisfy the condition

INSERT

- In its simplest form, it is used to add one or more tuples to a relation
- Attribute values should be listed in the same order as the attributes were specified in the **CREATE TABLE** command
- Constraints on data types are observed automatically
- Any integrity constraints as a part of the DDL specification are enforced

THE INSERT COMMAND

- Specify the relation name and a list of values for the tuple. All values including nulls are supplied.

```
U1:   INSERT INTO   EMPLOYEE
      VALUES        ( 'Richard', 'K', 'Marini', '653298653', '1962-12-30', '98
                      Oak Forest, Katy, TX', 'M', 37000, '653298653', 4 );
```

- The variation below inserts multiple tuples where a new table is loaded values from the result of a query.

```
insert into
Employees(EMPLOYEE_ID,EMPLOYEE_NAME,DEPARTMENT_I
D,SSN)
select SSN,FNAME,DNO,SUPER_SSN from EMPLOYEES1 where
SSN=4;
```

BULK LOADING OF TABLES

- Another variation of **INSERT** is used for bulk-loading of several tuples into tables
- A new table TNEW can be created with the same attributes as T and using LIKE and DATA in the syntax, it can be loaded with entire data.

```
CREATE TABLE D6EMPS AS
SELECT * FROM EMPLOYEES1 where SSN=4;
```

DELETE

- Removes tuples from a relation
  - Includes a WHERE-clause to select the tuples to be deleted
  - Referential integrity should be enforced
  - 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

THE DELETE COMMAND

- Removes tuples from a relation
  - Includes a `WHERE` clause to select the tuples to be deleted. The number of tuples deleted will vary.

```
U4A:  DELETE FROM  EMPLOYEE
      WHERE        Lname='Brown';

U4B:  DELETE FROM  EMPLOYEE
      WHERE        Ssn='123456789';

U4C:  DELETE FROM  EMPLOYEE
      WHERE        Dno=5;

U4D:  DELETE FROM  EMPLOYEE;
```

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 specified as part of DDL specification is enforced

UPDATE (CONTD.)

- Example: Change the location and controlling department number of project number 10 to 'Bellaire' and 5, respectively

```
UPDATE  PROJECT SET PLOCATION = 'Bellaire', DNUM = 5 WHERE
PNUMBER=10
```