Structured Query Language

Chapter 6

Contents

Contents

Introduction of Structured Query Language

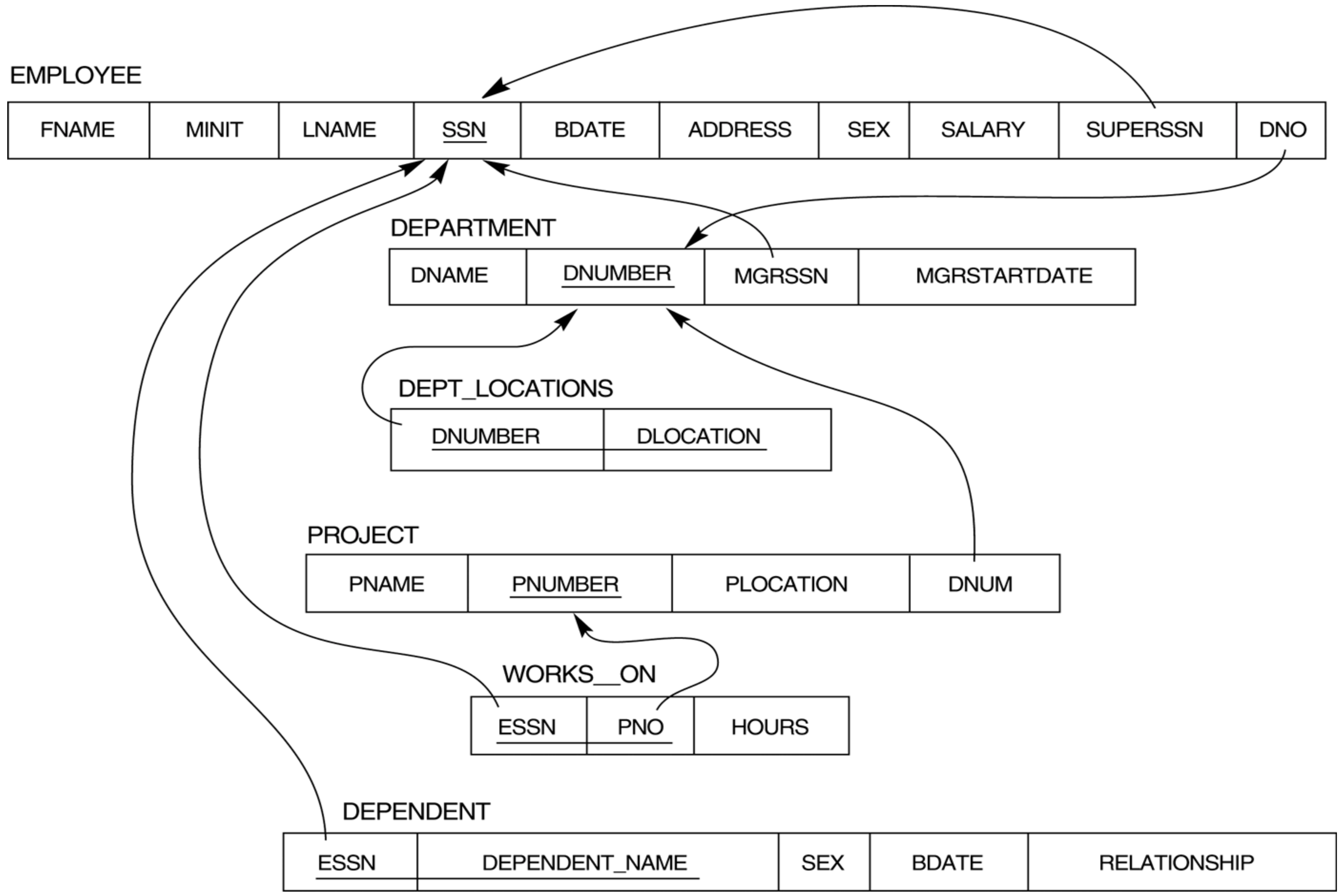
- **Structured Query Language** (SQL) is a standard computer language for relational database management and data manipulation.
- Basic SQL:
  - Data Definition Language (DDL)
    - Create, Alter, Drop
  - Data Manipulation Language (DML)
    - Select, Insert, Update, Delete
  - Data Control Language (DCL)
    - Commit, Rollback, Grant, Revoke

Contents

Data Definition Language (DDL)

- Permits specification of data types, structures and any data constraints
- All specifications are stored in the database
- Includes:
  - **CREATE**: make a new database object (database, table, index, user, stored query, …)
  - **ALTER**: modify an existing database object
  - **DROP**: destroy an existing database object

The COMPANY Database



Schema and Catalog Concepts in SQL

- **Schema**: a group of tables and other constructs that belong to the same database application

CREATE SCHEMA *Schema_Name*
AUTHORIZATION *Authorization_Identifier*;

CREATE SCHEMA Company **AUTHORIZATION** JSmith;

- **Catalog**: a named collection of schemas

CREATE TABLE

```
CREATE TABLE [SchemaName.]TableName
{(colName dataType [NOT NULL] [UNIQUE] [PRIMARY KEY]
[DEFAULT defaultOption]
[CHECK searchCondition] [,...]}
[PRIMARY KEY (listOfColumns),]
{[UNIQUE (listOfColumns),] [...,]}
{[FOREIGN KEY (listOfFKColumns)
 REFERENCES ParentTableName [(listOfCKColumns)]
 [ON UPDATE referentialAction]
 [ON DELETE referentialAction ]] [,...]}
{[CHECK (searchCondition)] [,...] })
```

CREATE TABLE

- **Base tables** (base relations)
  - Relation and its tuples are actually created and stored as a file by the DBMS.
- **Virtual relations**
  - Created through the CREATE VIEW statement.
- Some foreign keys may cause errors
  - Circular references
  - refer to a table that has not yet been created

Basic Data Types

- **Numeric data types**:
  - Integer numbers: INTEGER, INT, and SMALLINT
  - Floating-point (real) numbers: FLOAT or REAL, and DOUBLE PRECISION
- **Character-string data types**:
  - Fixed length: CHAR(n), CHARACTER(n)
  - Varying length: VARCHAR(n), CHAR VARYING(n), CHARACTER VARYING(n)
- **Bit-string data types**:
  - Fixed length: BIT(n)
  - Varying length: BIT VARYING(n)
- **Boolean data type**:
  - Values of TRUE or FALSE or NULL
- **Date-Time data types**:
  - Date components: YEAR, MONTH, and DAY (‘YYYY-MM-DD’)
  - Time components: HOUR, MINUTE, and SECOND (‘HH:MM:SS’)

Basic Data Types

- Additional data types
  - **Timestamp data type (TIMESTAMP)**
    - Includes the DATE and TIME fields
    - Plus a minimum of six positions for decimal fractions of seconds
    - Optional WITH TIME ZONE qualifier
  - **INTERVAL data type**
    - Specifies a relative value that can be used to increment or decrement an absolute value of a date, time, or timestamp

Domains

- Name used with the attribute specification
- Makes it easier to change the data type for a domain that is used by numerous attributes
- Improves schema readability

CREATE DOMAIN *DomainName* AS *DataType*
[CHECK *conditions*];

CREATE DOMAIN SSN_TYPE **AS** CHAR(9);
CREATE DOMAIN D_NUM **AS** INTEGER
CHECK (D_NUM>0 AND D_NUM<21);

Specifying Constraints

- Basic constraints:
  - Key and referential integrity constraints
  - Attribute constraints
  - Constraints on individual tuples within a relation

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**

PRIMARY KEY (Dnumber, DLocation)

- **UNIQUE** clause: Specifies alternate (secondary) keys.

Dname VARCHAR(15) **UNIQUE**;

Key and Referential Integrity Constraints

- **FOREIGN KEY** clause

FOREIGN KEY (*listOfFKColumns*)
REFERENCES *ParentTableName* [(*listOfCKColumns*)]
[ON UPDATE *referentialAction*]
[ON DELETE *referentialAction*]

- Referential triggered actions: RESTRICT (default), SET NULL, CASCADE, and SET DEFAULT

FOREIGN KEY Dno **REFERENCES** Department(Dnumber)
ON DELETE CASCADE
ON UPDATE CASCADE

Attribute Constraints

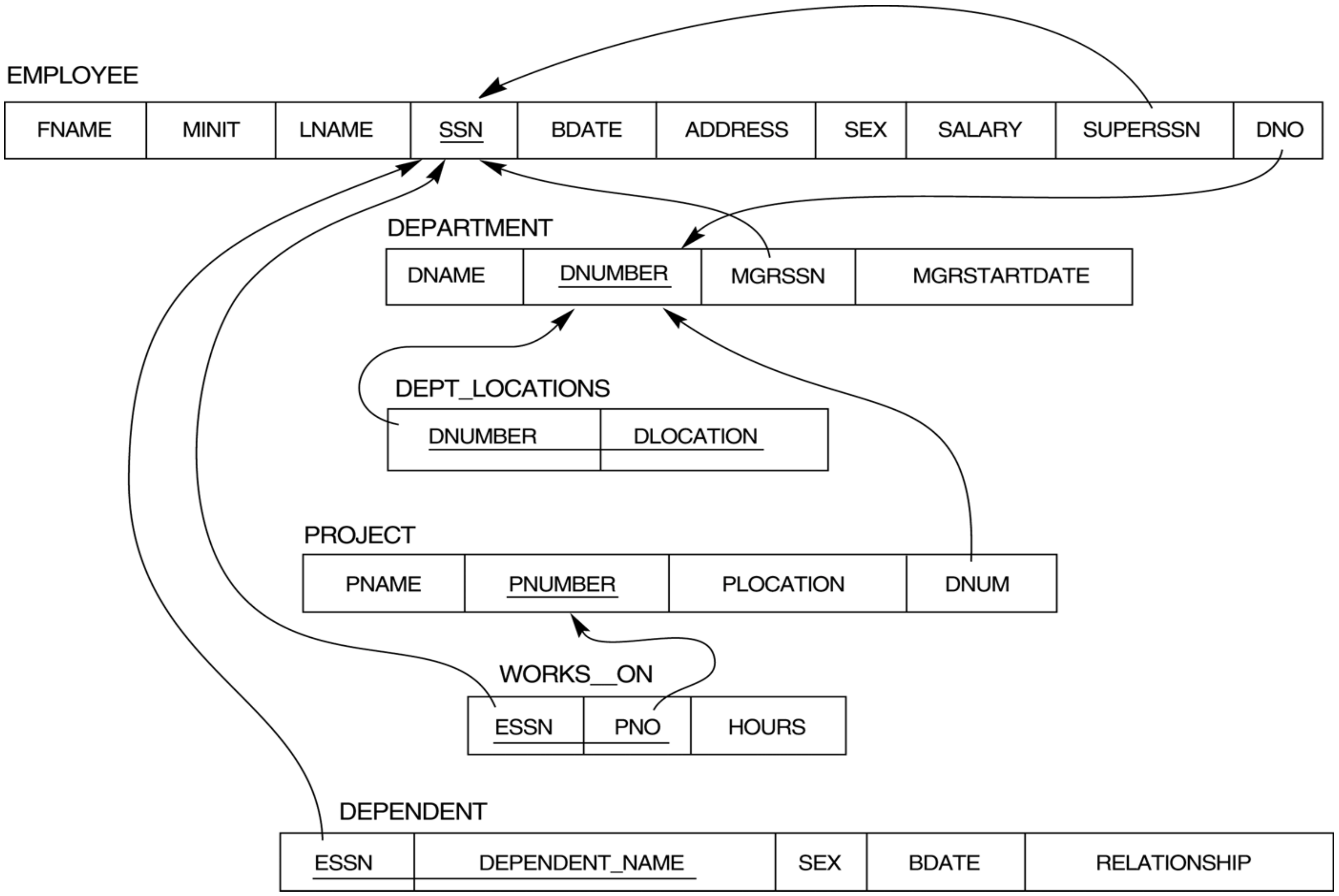
- NOT NULL
  - NULL is **not** permitted for a particular attribute
- Default values
  - DEFAULT <value> can be specified for an attribute
  - If no default clause is specified, the default value is NULL for attributes that do not have the NOT NULL constraint

Dno INT **NOT NULL DEFAULT 1**

- CHECK clause:

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

The COMPANY Database



```
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) );
```

```
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) );
```

Specifying Constraints

- Giving names to constraints
  - This is optional.
  - Keyword **CONSTRAINT**
  - The name is unique within a particular DB schema.
  - Used to identify a particular constraint in case it must be dropped later and replaced with another one.

```
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);
```

Constraints on individual tuples within a relation

- Specifying constraints on tuples using CHECK
  - Affected on each tuple individually as being inserted or modified (tuple-based constraints)
  - Ex: Department's create-date must be earlier than the manager's start-date:

```
CHECK (DEPT_CREATE_DATE < MGRSTARTDATE);
```

- More general constraints: CREATE ASSERTION

DROP Command

- Used to drop named schema elements: tables, domains, constraints, and the schema itself
- Drop behavior options:
  - CASCADE and RESTRICT

DROP SCHEMA Company **CASCADE;**

Or

DROP SCHEMA Company **RESTRICT;**

DROP Command

- Drop a table:

DROP TABLE Department **CASCADE**;

  - RESTRICT (default): dropped on if it is not referenced in any constraints or views
  - CASCADE: all such constraints and views that reference the table are dropped automatically from the schema along with the table itself
- Similarly, we can drop constraints & domains

ALTER Command

- ALTER command: change the definition of a base table or of other named schema elements
- Base tables: adding or dropping a column or constraints, changing a column definition.

```
ALTER TABLE Employee ADD Job VARCHAR(15);

ALTER TABLE Employee
        DROP COLUMN Address CASCADE;

ALTER TABLE Department
        ALTER COLUMN Mgr_ssn SET DEFAULT ‘333445555’;

ALTER TABLE Employee
        DROP CONSTRAINT Empsuperfk CASCADE;
```

Contents

SELECT Command

- SELECT command: retrieve information from a database
- SELECT command in SQL is the same as the SELECT operation in relational algebra.
- SQL allows a table (relation) to have two or more tuples that are identical in all their attribute values
- 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

SELECT Command

- Basic form:

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

- <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

SELECT Command

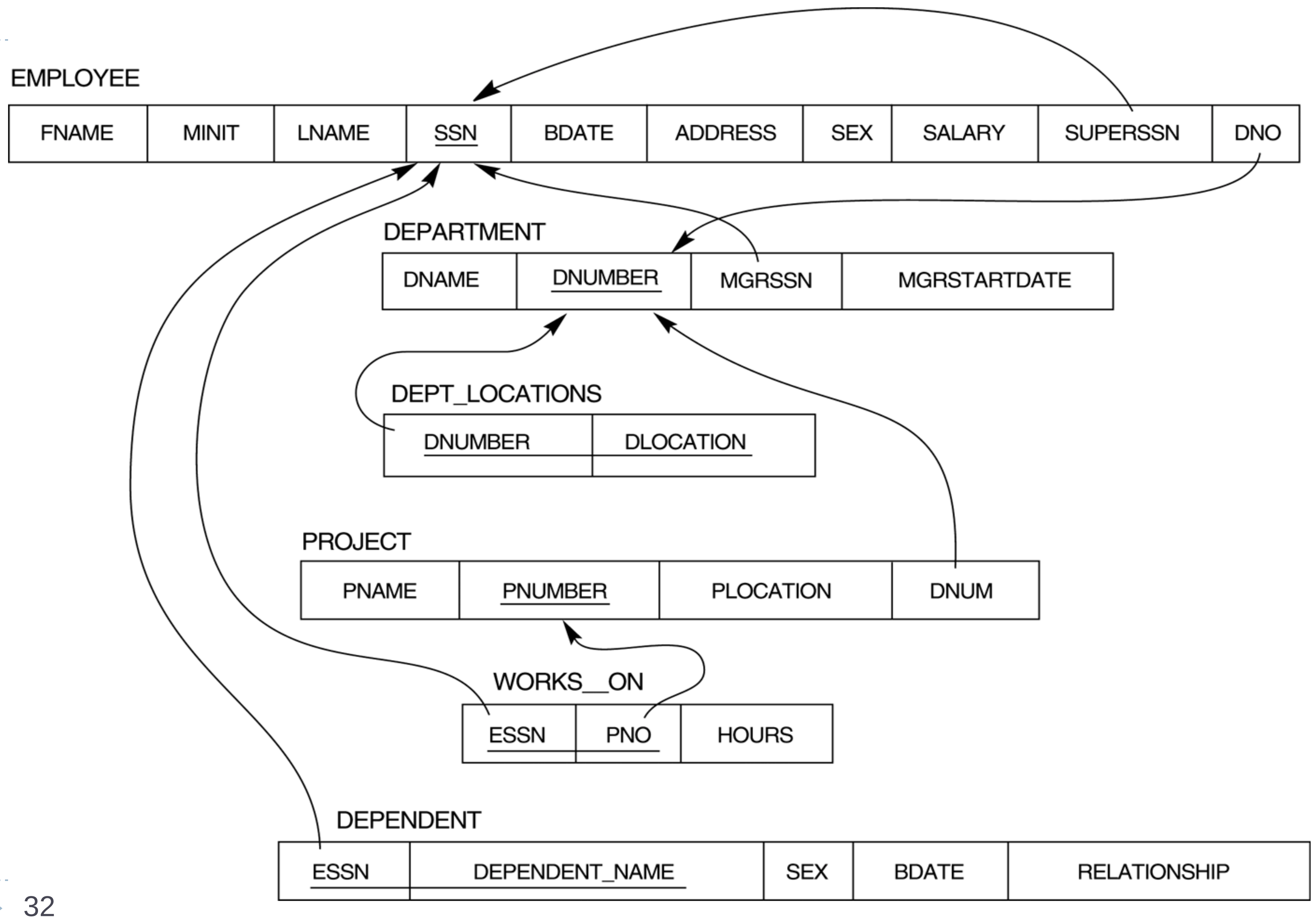
- Complete form:

```
SELECT [DISTINCT | ALL]
        {* | [columnExpression [AS newName]] [,...] }
FROM TableName [alias] [, ...]
[WHERE condition]
[GROUP BY columnList]    [HAVING condition]
[ORDER BY columnList]
```

SELECT Command

SELECT	: Specifies which columns are to appear in output
FROM	: Specifies table(s) to be used
WHERE	: Filters rows
GROUP BY	: Forms groups of rows with same column value
HAVING	: Filters groups subject to some condition
ORDER BY	: Specifies the order of the output

The COMPANY Database



SELECT Command

- Basic SQL queries: using the SELECT, PROJECT, and JOIN operations of the relational algebra

Query 0: Retrieve the birthdate and address of the employee whose name is 'John B. Smith'.

Q0:

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

- Similar to a SELECT-PROJECT pair of relational algebra operations:
  - SELECT clause specifies the projection attributes
  - WHERE clause specifies the selection condition
  - However, the result of the query may contain duplicate tuples

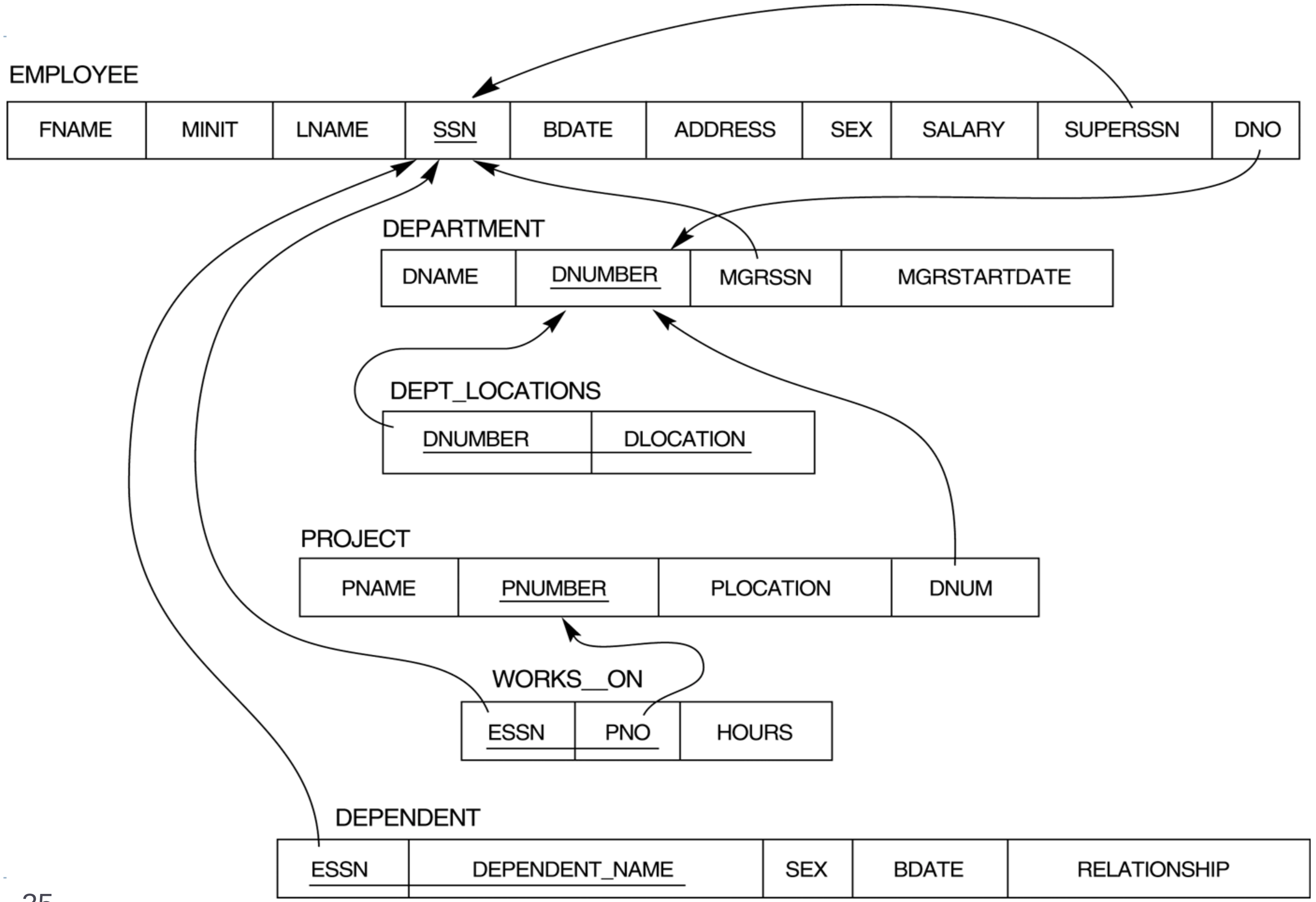
SELECT Command

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;
```

- Similar to a SELECT-PROJECT-JOIN sequence of relational algebra operations
  - (DNAME='Research'): selection condition (SELECT operation in relational algebra)
  - (DNUMBER=DNO): join condition (JOIN operation in relational algebra)

The COMPANY Database



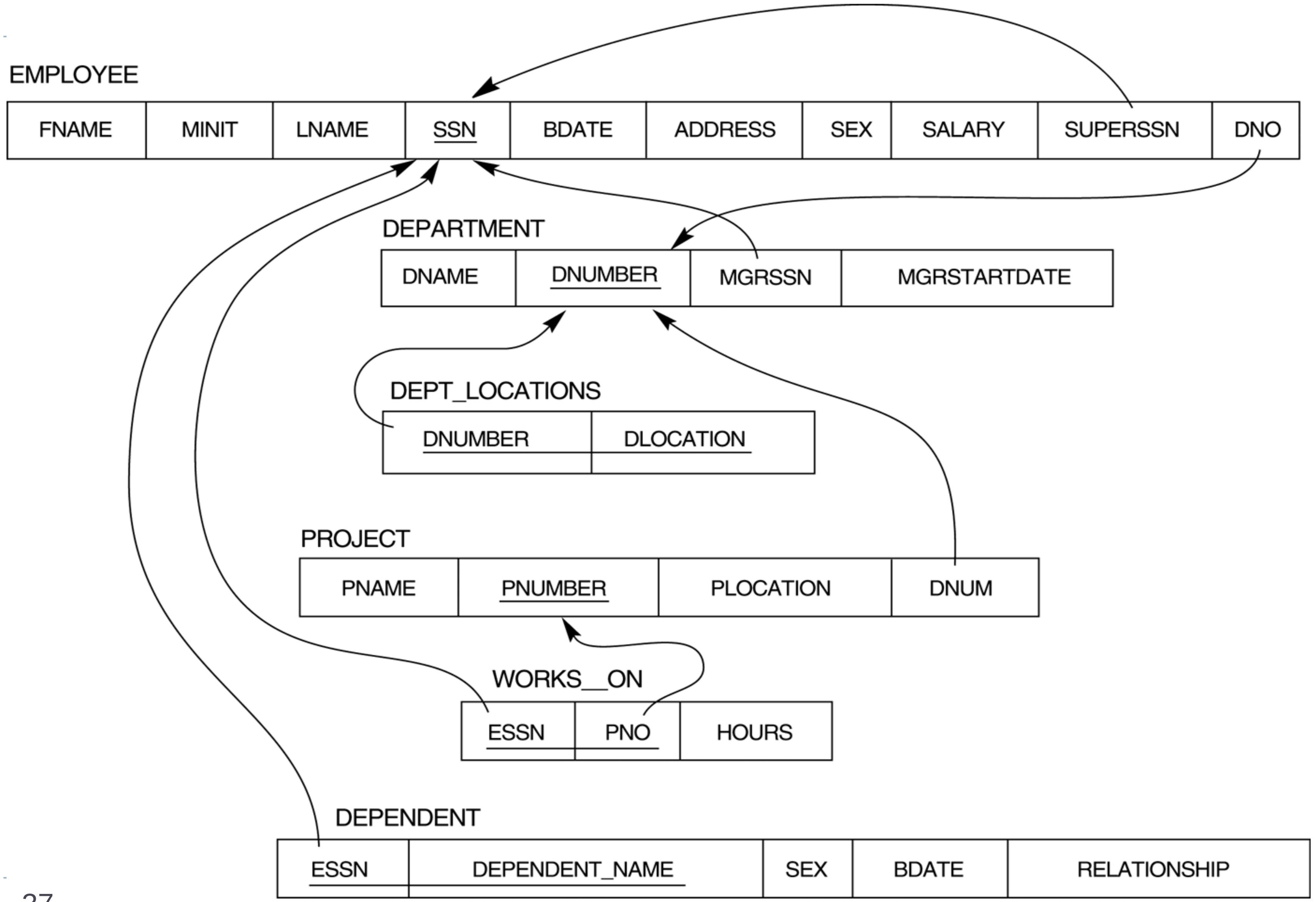
SELECT Command

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, Dnum, Lname, Bdate, Address
      FROM    Project, Department, Employee
      WHERE   Dnum = Dnumber AND MgrSSN = SSN
              AND Plocation='Stafford‘;
```

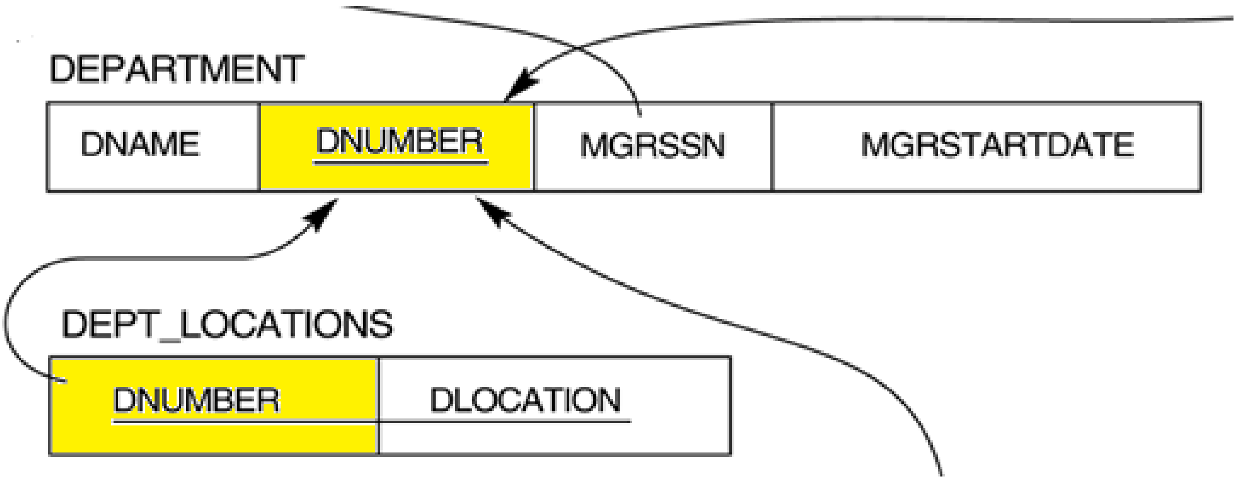
- Two join conditions:
  - Dnum = Dnumber: relates a project to its controlling department
  - MgrSSN = SSN: relates the controlling department to the employee who manages that department

The COMPANY Database



Ambiguous Attribute Names

- In SQL, we can use the same name for attributes as long as the attributes are in *different relations*. Query referring to attributes with the same name must *qualify* the attribute name with the relation name by *prefixing* the relation name to the attribute name
- Examples:
  - DEPARTMENT.DNUMBER and DEPT_LOCATIONS.DNUMBER



Aliases

- Some queries need to refer to the same relation twice: aliases are given to the relation name

Query 3: For each employee, retrieve the employee's name, and the name of his or her immediate supervisor.

Q3a:
```
SELECT   E.Fname, E.Lname, S.Fname, S.Lname
FROM     Employee E S
WHERE    E.SuperSSN = S.SSN;
```

- E and S are called aliases or tuple variables for the Employee relation
  - E represents employees in role of supervisees
  - S represents employees in role of supervisors

Aliases

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

Q3b: **SELECT** E.Fname, E.Lname, S.Fname, S.Lname
FROM Employee **AS** E, Employee **AS** S
WHERE E.SuperSSN = S.SSN;

- Renaming using aliases:

Employee **AS** E(FN, M, LN, SSN, BD, Addr, Sex, Sal, SSSN, DNO)

Unspecified WHERE-clause

- A missing WHERE-clause indicates no condition: all tuples of the relations in the FROM-clause are selected
- This is equivalent to the condition WHERE TRUE

Query 4: Retrieve the SSN values for all employees

Q4:
```
SELECT   SSN
FROM     Employee;
```

Unspecified WHERE-clause

- 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

Query 5: retrieve all combinations of Employee.SSN and Department.Dname

```
Q5:   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

Use of ASTERISK (*)

- An asterisk (*) stands for *all the attributes*

Query 6: retrieves all the attribute values of any Employee who works in Department number 5

```
Q6:   SELECT   *
      FROM     Employee
      WHERE    DNO = 5;
```

Query 7: retrieves all the attributes of an Employee and the attributes of the Department in which he or she works for every employee of the 'Research' department

```
Q7:   SELECT   *
      FROM     Employee, Department
      WHERE    Dname = 'Research' AND DNO = Dnumber;
```

Use of DISTINCT

- SQL does not treat a relation as a set: duplicate tuples can appear in a query result.
- To eliminate duplicate tuples, use the keyword DISTINCT

Query 8: Retrieve the salary of every employee (Q8A) and all distinct salary values (Q8B)

```
Q8a:   SELECT   Salary
       FROM     Employee;

Q8b:   SELECT   DISTINCT Salary
       FROM     Employee;
```

- The result of Q8A may have duplicate SALARY values, but Q8B's

Set Operations

- Set union **(UNION)**, set difference (**EXCEPT)** and set 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*
- UNION ALL, EXCEPT ALL, INTERSECT ALL

Set Operations

Query 9: 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.

Q10:
```
(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');
```

Substring pattern matching and arithmetic operators

- Two reserved characters: % and _

Query 10: Retrieve all employees whose address is in Houston, Texas.

```
Q10:  SELECT  *
      FROM    Employee
      WHERE   Address LIKE ‘%Houston,TX%’;
```

Query 11: Retrieve all employees whose SSN has ’88’ at the end.

```
Q11:  SELECT  *
      FROM    Employee
      WHERE   SSN LIKE ‘_ _ _ _ _ _ _ _88’;
```

Substring pattern matching and arithmetic operators

- Standard arithmetic operators: +, -, *, /

Query 12: show the resulting salaries if every employee working on "ProductX" is given 10% raise

Q12:
```
SELECT   Fname, Lname, 1.1*Salary AS INC_SAL
FROM     Employee, Works_on, Project
WHERE    SSN = ESSN AND PNO = Pnumber
         AND Pname = 'ProductX';
```

NULL & 3-valued logic

AND	True	False	Unknown
True	T	F	U
False	F	F	F
Unknown	U	F	U

NOT	
True	F
False	T
Unknown	U

OR	True	False	Unknown
True	T	T	T
False	T	F	U
Unknown	T	U	U

SELECT * **FROM** Employee **WHERE** SuperSSN IS NULL;

SELECT * **FROM** Employee **WHERE** SuperSSN IS NOT NULL;

SELECT Command

```
SELECT [DISTINCT | ALL]
        {* | [columnExpression [AS newName]] [,...] }
FROM TableName [alias] [, ...]
[WHERE condition]
[GROUP BY columnList]    [HAVING condition]
[ORDER BY columnList]
```

Nested Queries

- Complete SELECT-FROM-WHERE blocks within WHERE clause of another query
- Comparison operator IN
  - Compares value *v* with a set (or multiset) of values *V*
  - Evaluates to *TRUE* if *v* is one of the elements in *V*

Query 13: Retrieve the name and address of all employees who work for the 'Research' department

Q13:
```
SELECT   Fname, Lname, Address
FROM     Employee
WHERE    Dno IN ( SELECT  Dnumber
                  FROM    Department
                  WHERE   Dname = 'Research' );
```

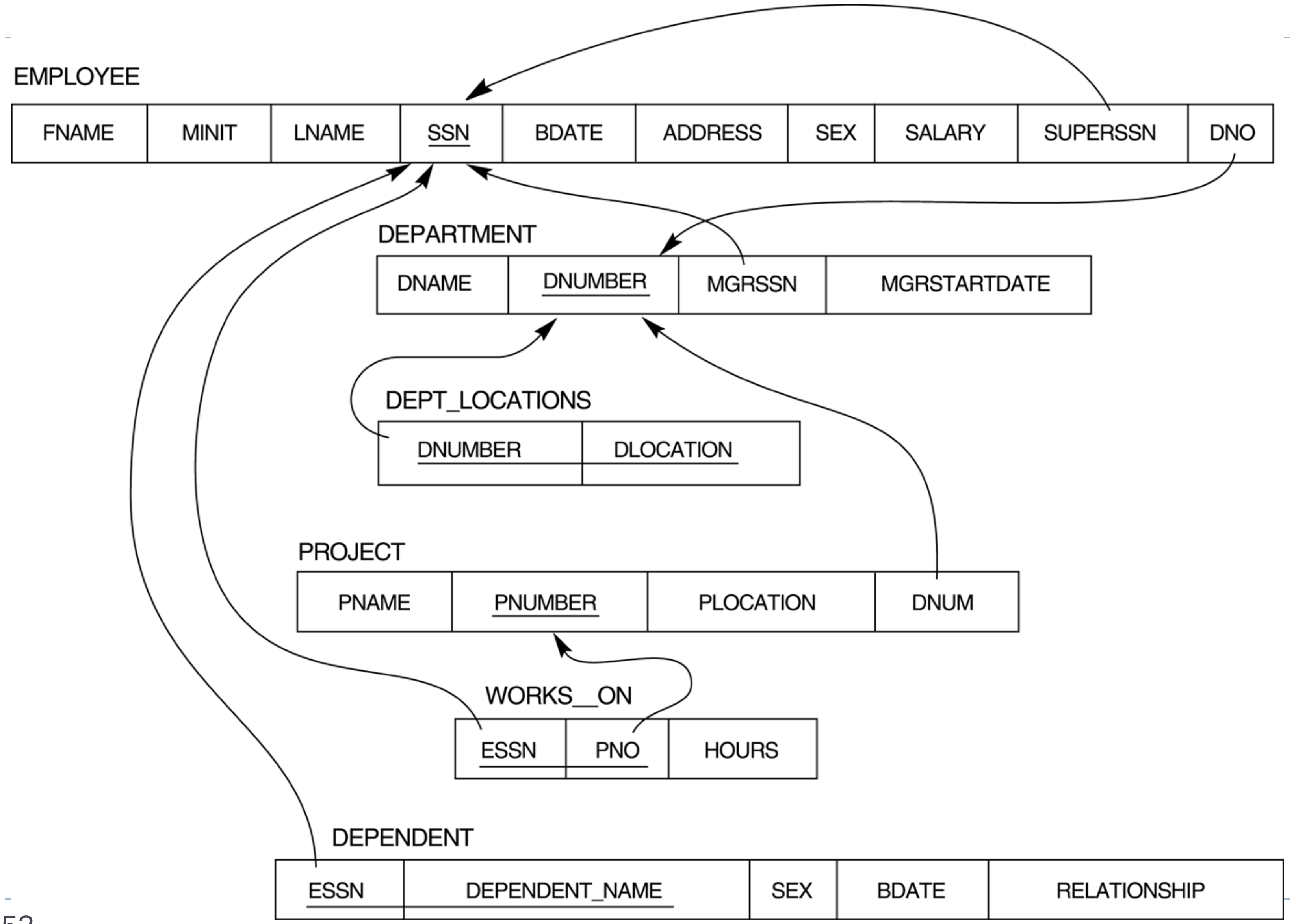
Correlated Nested Queries

- 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**

Query 14: Retrieve the name of each employee who has a dependent with the same first name as the employee.

```
Q14:  SELECT   E.Fname, E.Lname
      FROM     Employee E
      WHERE    E.SSN IN ( SELECT  ESSN
                          FROM    Dependent
                          WHERE   ESSN = E.SSN AND
                              E.Fname = Dependent_name);
```

The COMPANY Database



Correlated Nested Queries

- A query written with nested SELECT-FROM-WHERE blocks and using IN comparison operator can always be expressed as a single block query
- For example, Q14 may be written as in Q14A

```
Q14a: SELECT   E.Fname, E.Lname
      FROM     Employee E, Dependent D
      WHERE    E.SSN = D.ESSN AND
               E.Fname = D.Dependent_name;
```

Nested Query Exercises

Query 15: Retrieve the SSNs of all employees who work the same (project, hours) combination on some project that employee John Smith (SSN=123456789) works on (using a nested query)

Q15:

```
SELECT   DISTINCT  ESSN
FROM     Works_on
WHERE    (PNO, Hours) IN
                        ( SELECT PNO, Hours
                          FROM   Works_on
                          WHERE ESSN = '123456789' );
```

More Comparison Operators

- Operators that can be combined with ANY (or SOME), ALL: =, >, >=, <, <=, and <>

Query 16: Retrieve all employees whose salary is greater than the salary of all employees in department 5

```
Q16:  SELECT   *
      FROM     Employee
      WHERE    Salary > ALL ( SELECT Salary
                              FROM   Employee
                              WHERE  DNO=5 );
```

EXISTS and UNIQUE Functions

- **EXISTS** and **NOT EXISTS** function
  - Typically used in conjunction with a correlated nested query
  - EXISTS(Q) returns TRUE if the result of a query Q is NOT empty (Some tuples EXIST in the result).
  - NOT EXISTS(Q) returns TRUE if the result of a query Q is empty (No tuples are in the result).
- **UNIQUE(Q)** function
  - Returns TRUE if there are no duplicate tuples in the result of query Q

EXISTS Function

Query 14: Retrieve the name of each employee who has a dependent with the same first name as the employee

***Q14b*:**
```
SELECT   Fname, Lname
FROM     Employee
WHERE    EXISTS ( SELECT  *
                  FROM    Dependent
                  WHERE   ESSN = SSN AND
                          FName = Dependent_name);
```

EXISTS Function

Query 17: Retrieve the names of employees who have no dependents

Q17:
```
SELECT   Fname, Lname
FROM     Employee
WHERE    NOT EXISTS ( SELECT  *
                      FROM  Dependent
                      WHERE SSN = ESSN);
```

- In Q17, the correlated nested query retrieves all DEPENDENT tuples related to an EMPLOYEE tuple. If none exist , the EMPLOYEE tuple is selected

Enumerated Sets

- An explicit (enumerated) set of values in the WHERE-clause

Query 18: Retrieve the SSNs of all employees who work on project numbers 1, 2, or 3.

Q18:

```
SELECT  DISTINCT ESSN
FROM    Works_on
WHERE   PNO IN  (1, 2, 3);
```

Joined Relations

- Can specify a "joined relation" in the FROM-clause
- Allows the user to specify different types of joins
  - EQUIJOIN
  - NATURAL JOIN
  - LEFT OUTER JOIN
  - RIGHT OUTER JOIN
  - FULL OUTER JOIN

Joined Tables and Outer Joins

- Joined table
  - Permits users to specify a table resulting from a join operation in the FROM clause of a query

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

```
Q1a:   SELECT   Fname, Lname, Address
       FROM (Employee JOIN Department ON Dno = Dnumber)
       WHERE  Dname = ‘Research’;


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

Joined Tables and Outer Joins

- Specify different types of join
  - NATURAL JOIN
  - Various types of OUTER JOIN
- NATURAL JOIN on two relations R and S
  - **No join condition specified**
  - Implicit EQUIJOIN condition for each pair of attributes with same name from R and S

Joined Tables in SQL and Outer Joins (cont'd.)

- **Inner join**
  - Default type of join in a joined table
  - Tuple is included in the result only if a matching tuple exists in the other relation
- **LEFT OUTER JOIN**
  - Every tuple in LEFT table must appear in result
  - If no matching tuple
    - Padded with NULL values for attributes of RIGHTtable

Joined Tables in SQL and Outer Joins (cont'd.)

- RIGHT OUTER JOIN
  - Every tuple in RIGHT table must appear in result
  - If no matching tuple
    - Padded with NULL values for the attributes of LEFT table
- FULL OUTER JOIN

Joined Relations - Examples

Query 3: For each employee, retrieve the employee's name, and the name of his or her immediate supervisor.

Q3a:
```
SELECT  E.Fname, E.Lname, S.Fname, S.Lname
FROM    Employee E S
WHERE   E.SuperSSN = S.SSN;
```

Q3c:
```
SELECT  E.Fname, E.Lname, S.Fname, S.Lname
FROM    ( Employee E LEFT OUTER JOIN
          Employee S ON  E.SuperSSN = S.SSN );
```

- **Compare two queries???**

Joined Relations - Examples

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;
```

- could be written as:

```
Q1a:   SELECT  Fname, Lname, Address
       FROM    (Employee JOIN Department ON Dnumber = Dno)
       WHERE   Dname = 'Research';

Q1b:   SELECT  Fname, Lname, Address
       FROM    (Employee NATURAL JOIN (Department
                       AS Dept(Dname, Dno, MSSN, MSDate)))
       WHERE   Dname = 'Research';
```

Joined Relations - Examples

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

Q2a:

```
SELECT  Pnumber, Dnum, Lname, Bdate, Address
FROM    ((Project JOIN Department ON Dnum =
           Dnumber) JOIN Employee ON MGRSSN = SSN))
WHERE   Plocation = 'Stafford' ;
```

AGGREGATE FUNCTIONS

- **COUNT, SUM, MAX, MIN, AVG**

Query 19: Find the max, min, & average salary among all employees

```
Q19:  SELECT  MAX(Salary), MIN(Salary), AVG(Salary)
      FROM    Employee;
```

AGGREGATE FUNCTIONS

Queries 20: Retrieve the total number of employees in the company

```
Q20:  SELECT  COUNT (*)
      FROM    Employee;
```

Queries 21: Retrieve the number of employees in the 'Research' department

```
Q21:  SELECT  COUNT (*)
      FROM    Employee, Department
      WHERE   Dno = Dnumber AND Dname = 'Research’;
```

- Note: NULL values are discarded wrt. aggregate functions as applied to a particular column

GROUPING

- A GROUP BY-clause is for specifying the grouping attributes, which must also appear in the SELECT-clause
- Each subgroup of tuples consists of the set of tuples that have the same value for the grouping attribute(s)
- 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 aggregate function is applied to each subgroup independently
- If NULLs exist in grouping attribute
  - Separate group created for all tuples with a NULL value in grouping attribute

SELECT Command

```
SELECT [DISTINCT | ALL]
        {* | [columnExpression [AS newName]] [,...] }
FROM TableName [alias] [, ...]
[WHERE condition]
[GROUP BY columnList]    [HAVING condition]
[ORDER BY columnList]
```

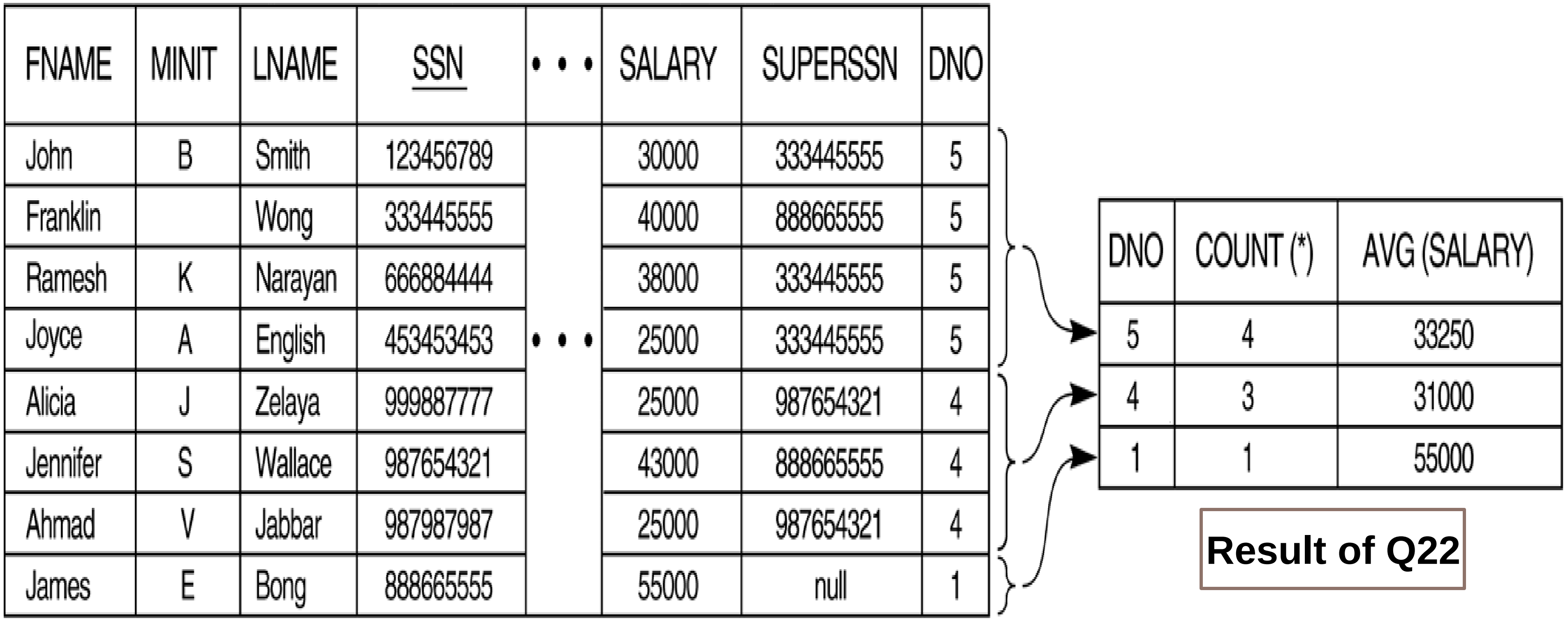
GROUPING

Query 22: For each department, retrieve the department number, the number of employees in the department, and their average salary

Q22:
```
SELECT      Dno, COUNT (*), AVG (Salary)
FROM        Employee
GROUP BY    Dno;
```

- In Q22, 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

GROUPING: Q22 result



FNAME	MINIT	LNAME	SSN	• • •	SALARY	SUPERSSN	DNO
John	B	Smith	123456789		30000	333445555	5
Franklin		Wong	333445555		40000	888665555	5
Ramesh	K	Narayan	666884444		38000	333445555	5
Joyce	A	English	453453453	• • •	25000	333445555	5
Alicia	J	Zelaya	999887777		25000	987654321	4
Jennifer	S	Wallace	987654321		43000	888665555	4
Ahmad	V	Jabbar	987987987		25000	987654321	4
James	E	Bong	888665555		55000	null	1

DNO	COUNT (*)	AVG (SALARY)
5	4	33250
4	3	31000
1	1	55000

Result of Q22

Grouping EMPLOYEE tuples by the value of DNO.

GROUPING: 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)

GROUPING: THE HAVING-CLAUSE

Query 23: 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.

Q23:

```
SELECT      Pnumber, Pname, COUNT (*)
FROM        Project, Works_on
WHERE       Pnumber = Pno
GROUP BY    Pnumber, Pname
HAVING      COUNT (*) > 2;
```

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 24: 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

Q24:

```
SELECT Dname, Lname, Fname, Pname
FROM    Department, Employee, Works_on, Project
WHERE  Dnumber = Dno AND SSN = ESSN
                    AND Pno = Pnumber
ORDER BY  Dname, Lname [DESC|ASC]
```

SELECT Command

```
SELECT [DISTINCT | ALL]
        {* | [columnExpression [AS newName]] [,...] }
FROM TableName [alias] [, ...]
[WHERE condition]
[GROUP BY columnList]    [HAVING condition]
[ORDER BY columnList]
```

SELECT Command

SELECT	Specifies which columns are to appear in output
FROM	Specifies table(s) to be used
WHERE	Filters rows
GROUP BY	Forms groups of rows with same column value
HAVING	Filters groups subject to some condition
ORDER BY	Specifies the order of the output

Contents

Insert Command

- 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

INSERT INTO *TableName (Attribute1, Attribute2, ...)*
VALUES (*value1, value2, ...*);

Insert Command

- Insert a tuple for a new EMPLOYEE:

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.

U2:

```
INSERT INTO Employee (Fname, Lname, SSN)
VALUES ('Richard', 'Marini', '653298653');
```

Insert Command

- Important note: Only the constraints specified in the DDL commands are automatically enforced by the DBMS when updates are applied to the database
- Another variation of INSERT allows insertion of multiple tuples resulting from a query into a relation

Insert Command

- 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 U3, and is loaded with the summary information retrieved from the database by the query in U3A*

```
U3:    CREATE TABLE  Depts_info
            ( Dept_name   VARCHAR(10),
              No_of_emps  INTEGER,
              Total_sal   INTEGER);

U3A:   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;
```

Delete Command

```
DELETE FROM TableName
WHERE Condition;
```

- Removes tuples from a relation
- 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

Delete Command - 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;
```

Update Command

```
UPDATE TableName
SET    Set-Clause
WHERE  Condition;
```

- 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

Update Command

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

U5:
```
UPDATE    Project
SET       Plocation = 'Bellaire', Dnum = 5
WHERE     Pnumber = 10;
```

Update Command

- Example: *Give all employees in the 'Research' department a 10% raise in salary.*

U6:

```
UPDATE   Employee
SET      Salary = Salary *1.1
WHERE    Dno  IN (SELECT  Dnumber
                  FROM  Department
                  WHERE Dname = 'Research');
```

Advanced DDL: Assertions & Triggers

- ASSERTIONs to express constraints that do not fit in the basic SQL categories
- Mechanism: CREATE ASSERTION
  - components include: a constraint name, followed by CHECK, followed by a condition

Advanced DDL: Assertions & Triggers

- Example: *The salary of an employee must not be greater than the salary of the manager of the department that the employee works for'*

```
CREATE ASSERTION Salary_constraint
CHECK (NOT EXISTS (SELECT *
                   FROM Employee E, Employee M,
                        Department D
                   WHERE E.Salary > M.Salary AND
                      E.Dno = D.Number AND
                      D.MGRSSN = M.SSN));
```

Advanced DDL: Assertions & Triggers

- Triggers: to specify the type of action to be taken as certain events occur and as certain conditions are satisfied
- Details of triggers: presentation and lab

Views

- A view is a “virtual” table that is derived from other tables
- Allows for limited update operations (since the table may not physically be stored)
- Allows full query operations
- A convenience for expressing certain operations

VIEWs

- Specify a different WORKS_ON table (view)

```
CREATE VIEW     Works_on_new AS
      SELECT    Fname, Lname, Pname, Hours
      FROM      Employee, Project, Works_on
      WHERE     SSN = ESSN AND Pno = Pnumber;
```

- We can specify SQL queries on a newly create table (view):

```
SELECT Fname, Lname From Works_on_new
WHERE Pname = ‘Seena’;
```

- When no longer needed, a view can be dropped:

```
DROP VIEW Works_on_new;
```

View Update and Inline Views

- Update on a view defined on a single table without any aggregate functions
  - Can be mapped to an update on underlying base table
- View involving joins
  - Often not possible for DBMS to determine which of the updates is intended

Contents

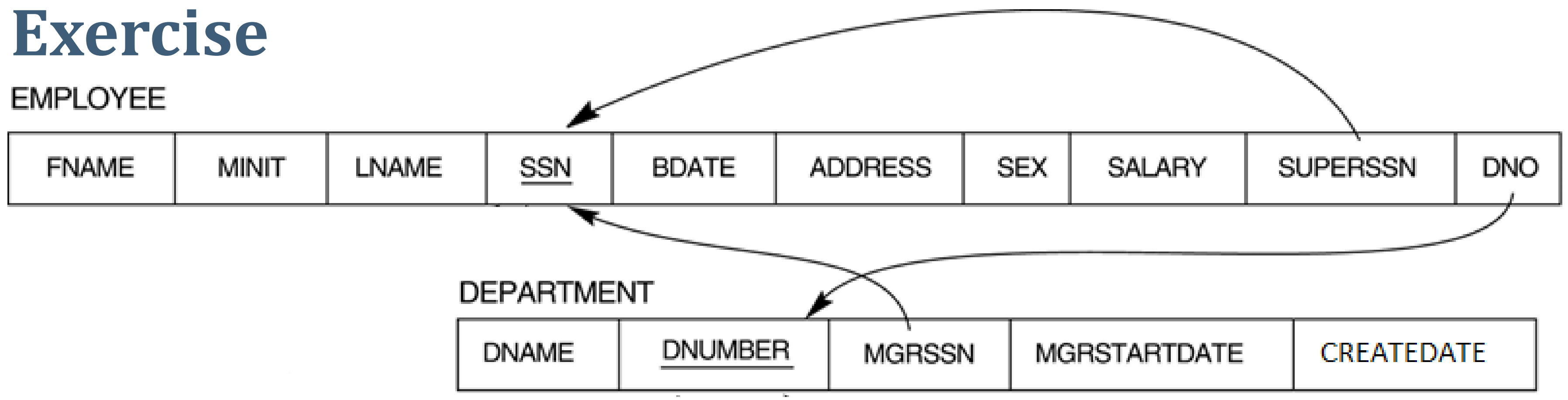
Lab

Summary

- SQL developments: an overview
- SQL
  - DDL: Create, Alter, Drop
  - DML: select, insert, update, delete
  - Introduction to advanced DDL (assertions & triggers), views, DCL (commit, rollback, grant, revoke)

Q
A

Exercise



EMPLOYEE:

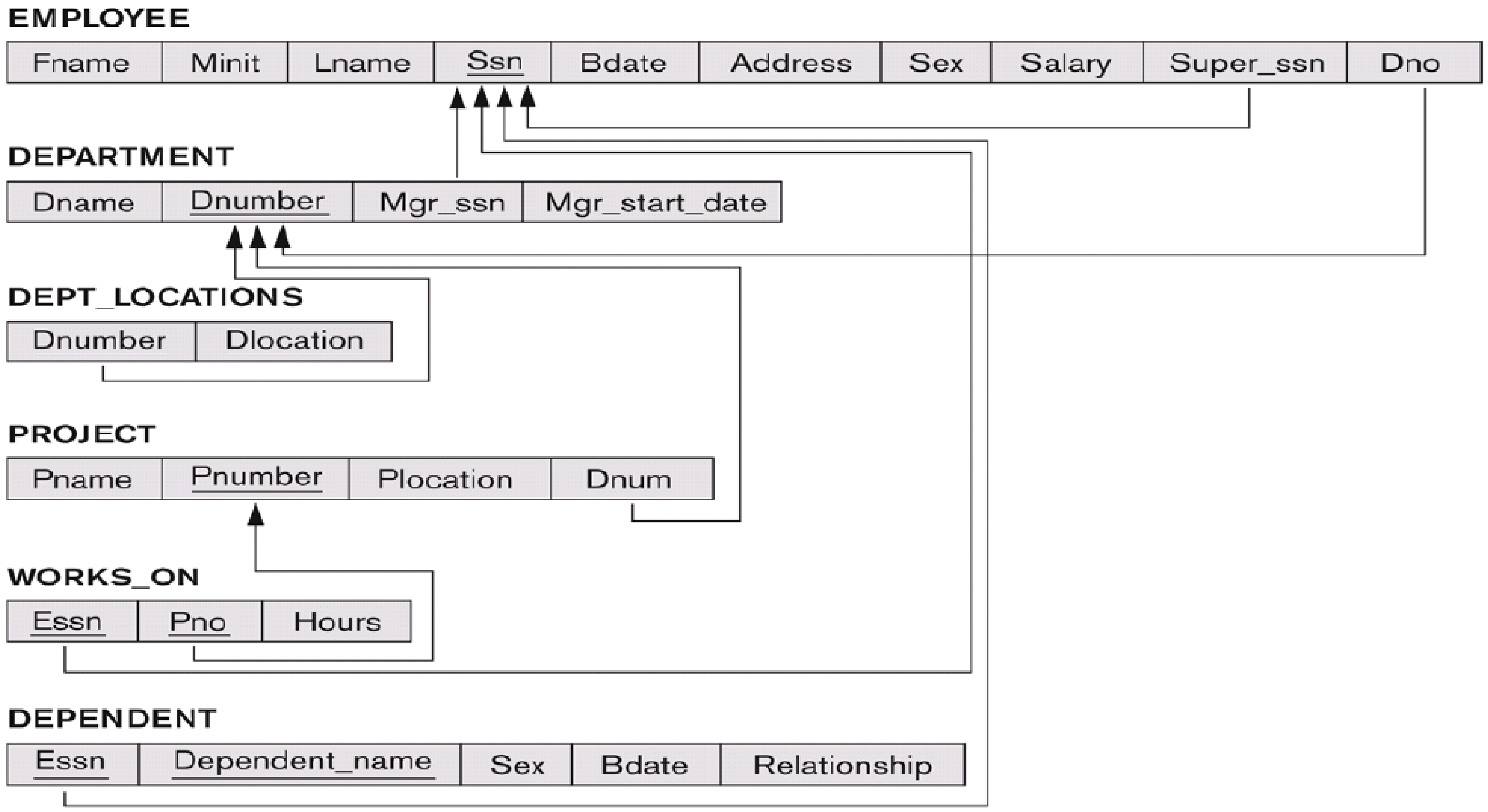
- Fname, Lname: VARCHAR(15), NOT NULL
- Minit: CHAR
- SSN: CHAR(9), NOT NULL, PRIMARY KEY
- Bdate: DATE, earlier than "1/1/1999"
- Address: VARCHAR(100)
- Sex: CHAR, {F/M}
- Salary: DECIMAL(10,2)
- SuperSSN: CHAR(9), refers to EMPLOYEE(SSN)
- Dno: INT, NOT NULL, default value = 1, refers to DEPARTMENT(Dnumber) – ON DELETE SET DEFAULT

DEPARTMENT:

- Dname: VARCHAR(15), NOT NULL, UNIQUE
- Dnumber: INT, NOT NULL, PRIMARY KEY
- MgrSSN: CHAR(9), NOT NULL, default value = '888665555', refers to EMPLOYEE(SSN) – ON DELETE SET DEFAUL, ON UPDATE CASCADE
- MgrStartDate: DATE
- CreateDate: DATE, earlier than MgrStartDate

CREATE TABLE

```
CREATE TABLE [SchemaName.]TableName
{(colName dataType [NOT NULL] [UNIQUE] [PRIMARY KEY]
[DEFAULT defaultOption]
[CHECK searchCondition] [,...]}
[PRIMARY KEY (listOfColumns),]
{[UNIQUE (listOfColumns),] [...,]}
{[FOREIGN KEY (listOfFKColumns)
  REFERENCES ParentTableName [(listOfCKColumns)]
  [ON UPDATE referentialAction]
  [ON DELETE referentialAction ]] [,...]}
{[CHECK (searchCondition)] [,...] })
```


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

Exercise 2

1. For each employee, retrieve the employee’s first name and last name and the first and last name of his/her immediate supervisor.
2. Retrieve the names of all employees in the departments which are located in Houston
3. List the names of all employees who have a dependent with the same first name as themselves
4. For each project, calculate the total number of employees who work for it, and the total number of hours that these employees work for the project.
5. Retrieve the average salary of all female employees.
6. For each department whose average employee salary is more than $30.000, retrieve the department name and the number of employees work for that department.