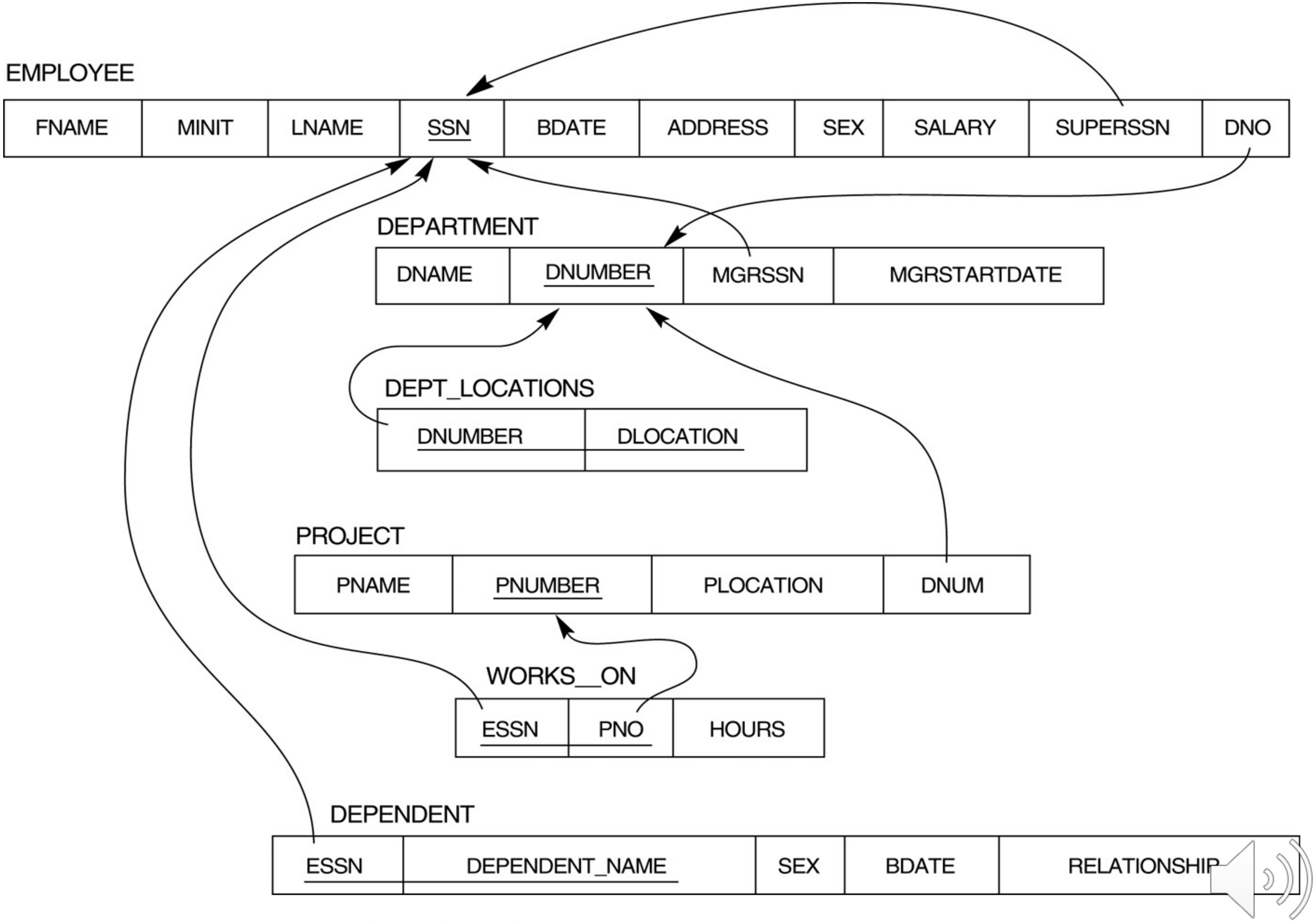
SQL
(Structured Query Language)

Assoc. Prof. Dr. Dang Tran Khanh

Outline

- The COMPANY Database
- SQL developments: an overview
- SQL
  - DDL: create, alter, drop
  - DML: select, insert, update, delete
  - DCL: commit, rollback, grant, revoke
- Reading Suggestion:
  - [1]: Chapters 6, 7
  - http://www.oracle.com

The COMPANY Database



EMPLOYEE

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO

DEPARTMENT

DNAME	DNUMBER	MGRSSN	MGRSTARTDATE

DEPT_LOCATIONS

DNUMBER	DLOCATION

PROJECT

PNAME	PNUMBER	PLOCATION	DNUM

WORKS__ON

ESSN	PNO	HOURS

DEPENDENT

ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP

SQL developments: an overview

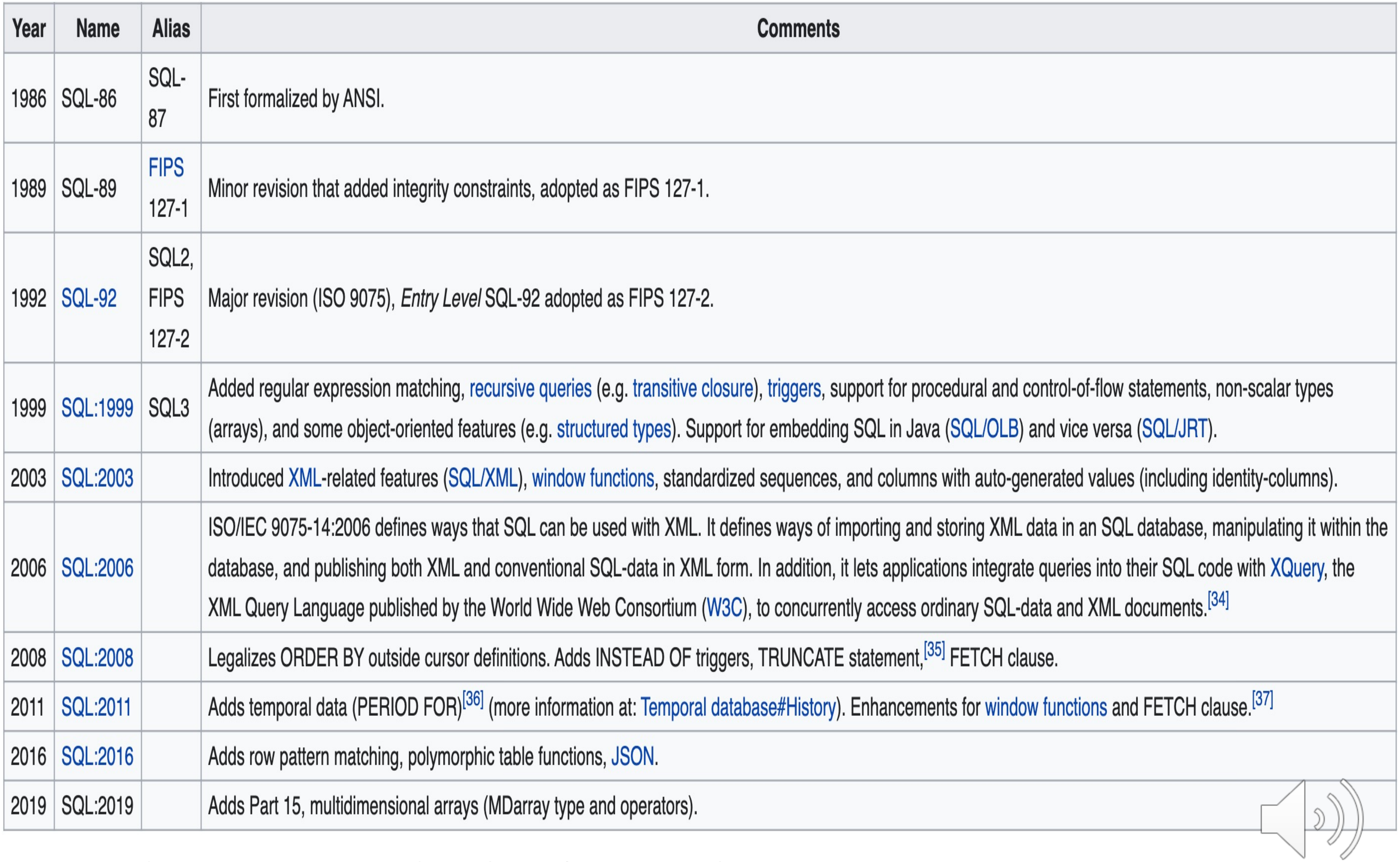
- In 1986, ANSI and ISO published an initial standard for SQL: SQL-86 or SQL1
- In 1992, first major revision to ISO standard occurred, referred to as SQL2 or SQL-92
- In 1999, SQL:1999 (SQL3) was released with support for recursive queries, triggers, procedural and control-of-flow statements, non-scalar types, and some object-oriented features
- In late 2003, SQL:2003 was released with XML-related features
- SQL:2006 was published with W3C XQuery support
- SQL:2008: INSTEAD OF triggers, TRUNCATE statement, etc.
- SQL:2011 was the 7th revision of the SQL database query language. It was formally adopted in December 2011

SQL developments: an overview

- SQL:2016: 44 new optional features. 22 belong to the JSON functionality, >10 are related to polymorphic table functions
- SQL:2019: Multi-dimensional arrays. It specifies a multidimensional array type (MDarray) for SQL. This part of the standard consists solely of optional features

SQL developments: an overview

(http://en.wikipedia.org/wiki/SQL)



Year	Name	Alias	Comments
1986	SQL-86	SQL-87	First formalized by ANSI.
1989	SQL-89	FIPS 127-1	Minor revision that added integrity constraints, adopted as FIPS 127-1.
1992	SQL-92	SQL2, FIPS 127-2	Major revision (ISO 9075), *Entry Level* SQL-92 adopted as FIPS 127-2.
1999	SQL:1999	SQL3	Added regular expression matching, recursive queries (e.g. transitive closure), triggers, support for procedural and control-of-flow statements, non-scalar types (arrays), and some object-oriented features (e.g. structured types). Support for embedding SQL in Java (SQL/OLB) and vice versa (SQL/JRT).
2003	SQL:2003		Introduced XML-related features (SQL/XML), window functions, standardized sequences, and columns with auto-generated values (including identity-columns).
2006	SQL:2006		ISO/IEC 9075-14:2006 defines ways that SQL can be used with XML. It defines ways of importing and storing XML data in an SQL database, manipulating it within the database, and publishing both XML and conventional SQL-data in XML form. In addition, it lets applications integrate queries into their SQL code with XQuery, the XML Query Language published by the World Wide Web Consortium (W3C), to concurrently access ordinary SQL-data and XML documents.[34]
2008	SQL:2008		Legalizes ORDER BY outside cursor definitions. Adds INSTEAD OF triggers, TRUNCATE statement,[35] FETCH clause.
2011	SQL:2011		Adds temporal data (PERIOD FOR)[36] (more information at: Temporal database#History). Enhancements for window functions and FETCH clause.[37]
2016	SQL:2016		Adds row pattern matching, polymorphic table functions, JSON.
2019	SQL:2019		Adds Part 15, multidimensional arrays (MDarray type and operators).

SQL developments: an overview

(http://en.wikipedia.org/wiki/SQL)

- SQL is designed for a specific purpose: to query data contained in a relational database. SQL is a set-based, declarative query language, not an imperative language such as C/C# or Java (for example)
- However, there are extensions to Standard SQL which add procedural programming language functionality

Source	Common Name	Full Name
ANSI/ISO Standard	SQL/PSM	SQL/Persistent Stored Modules
Interbase/ Firebird	PSQL	Procedural SQL
IBM	SQL PL	SQL Procedural Language (implements SQL/PSM)
Microsoft/ Sybase	T-SQL	Transact-SQL
Mimer SQL	SQL/PSM	SQL/Persistent Stored Module (implements SQL/PSM)
MySQL	SQL/PSM	SQL/Persistent Stored Module (implements SQL/PSM)
Oracle	PL/SQL	Procedural Language/SQL
PostgreSQL	PL/pgSQL	Procedural Language/PostgreSQL Structured Query Language (based on Oracle PL/SQL)
PostgreSQL	PL/PSM	Procedural Language/Persistent Stored Modules (implements SQL/PSM)

Outline

- The COMPANY Database
- SQL developments: an overview
- SQL
  - DDL: create, alter, drop
  - DML: select, insert, update, delete
  - DCL: commit, rollback, grant, revoke
- Reading Suggestion:
  - [1]: Chapters 6, 7
  - http://www.oracle.com

DDL: Create, Alter, Drop

CREATE SCHEMA

- CREATE SCHEMA SchemaName AUTHORIZATION AuthorizationIdentifier;
- To create a relational database schema: started with SQL-92

```
CREATE SCHEMA Company AUTHORIZATION JSmith;
```

DDL: Create, Alter, Drop

CREATE TABLE

- CREATE TABLE Company.TableName …

or

- CREATE TABLE TableName …

DDL: Create, Alter, Drop

CREATE TABLE

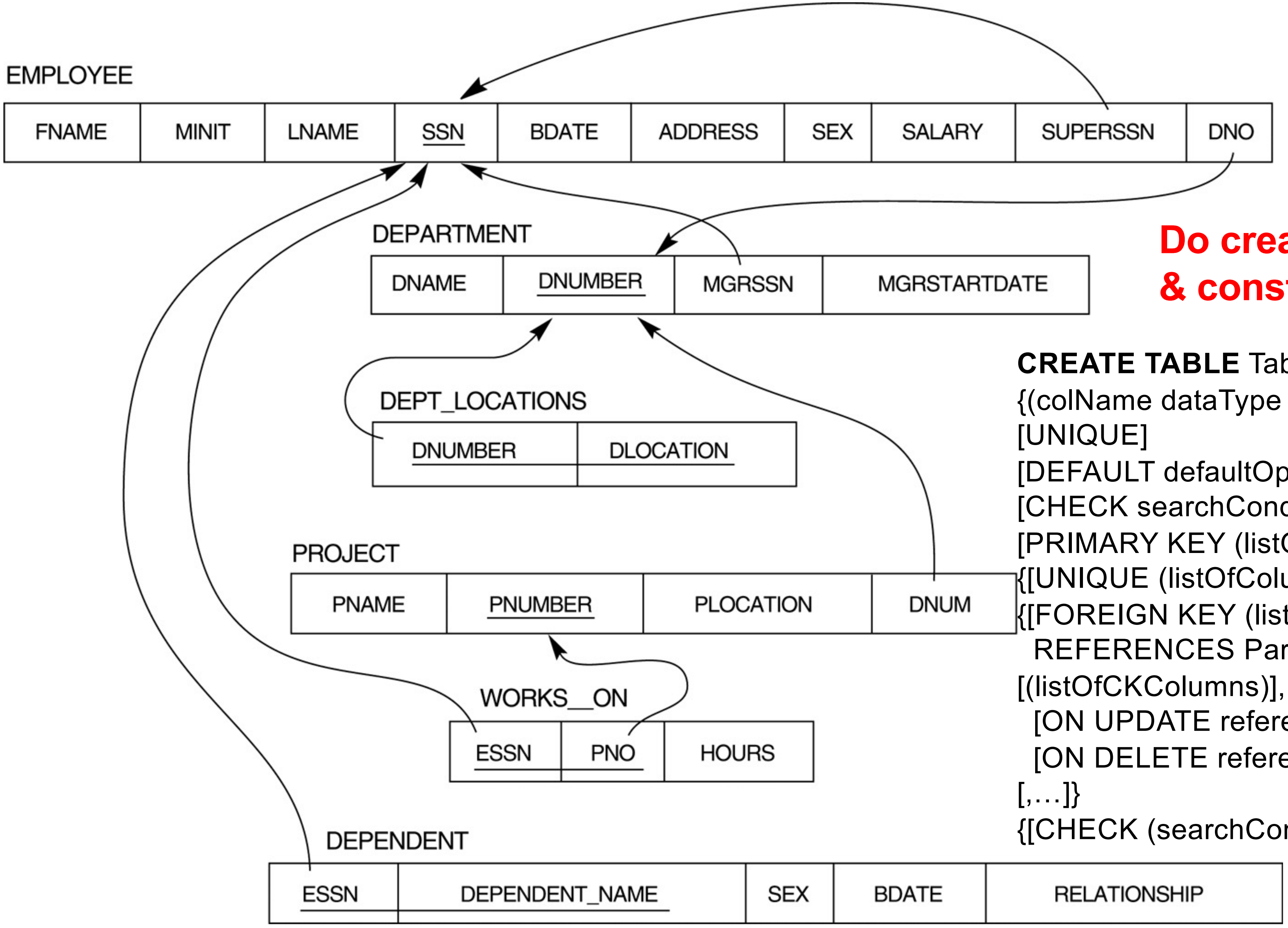
```
CREATE TABLE TableName
{(colName dataType [NOT NULL] [UNIQUE]
[DEFAULT defaultOption]
[CHECK searchCondition] [,...]}
[PRIMARY KEY (listOfColumns),]
{[UNIQUE (listOfColumns),] […,]}
{[FOREIGN KEY (listOfFKColumns)
  REFERENCES ParentTableName [(listOfCKColumns)],
  [ON UPDATE referentialAction]
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

DDL: Create, Alter, Drop

CREATE TABLE

- DataType
  - Numeric: INT or INTEGER, FLOAT or REAL, DOUBLE PRECISION, …
  - Character string: fixed length CHAR(n), varying length VARCHAR(n)
  - Bit string: BIT(n), e.g. B’1001’
  - Boolean: true, false or NULL
  - Date, Time: DATE ‘YYYY-MM-DD’, TIME ‘HH:MM:SS’
  - TIMESTAMP: date + time + …
- CREATE DOMAIN DomainName AS DataType [CHECK conditions];

The COMPANY Database



EMPLOYEE

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO

DEPARTMENT

DNAME	DNUMBER	MGRSSN	MGRSTARTDATE

DEPT_LOCATIONS

DNUMBER	DLOCATION

PROJECT

PNAME	PNUMBER	PLOCATION	DNUM

WORKS__ON

ESSN	PNO	HOURS

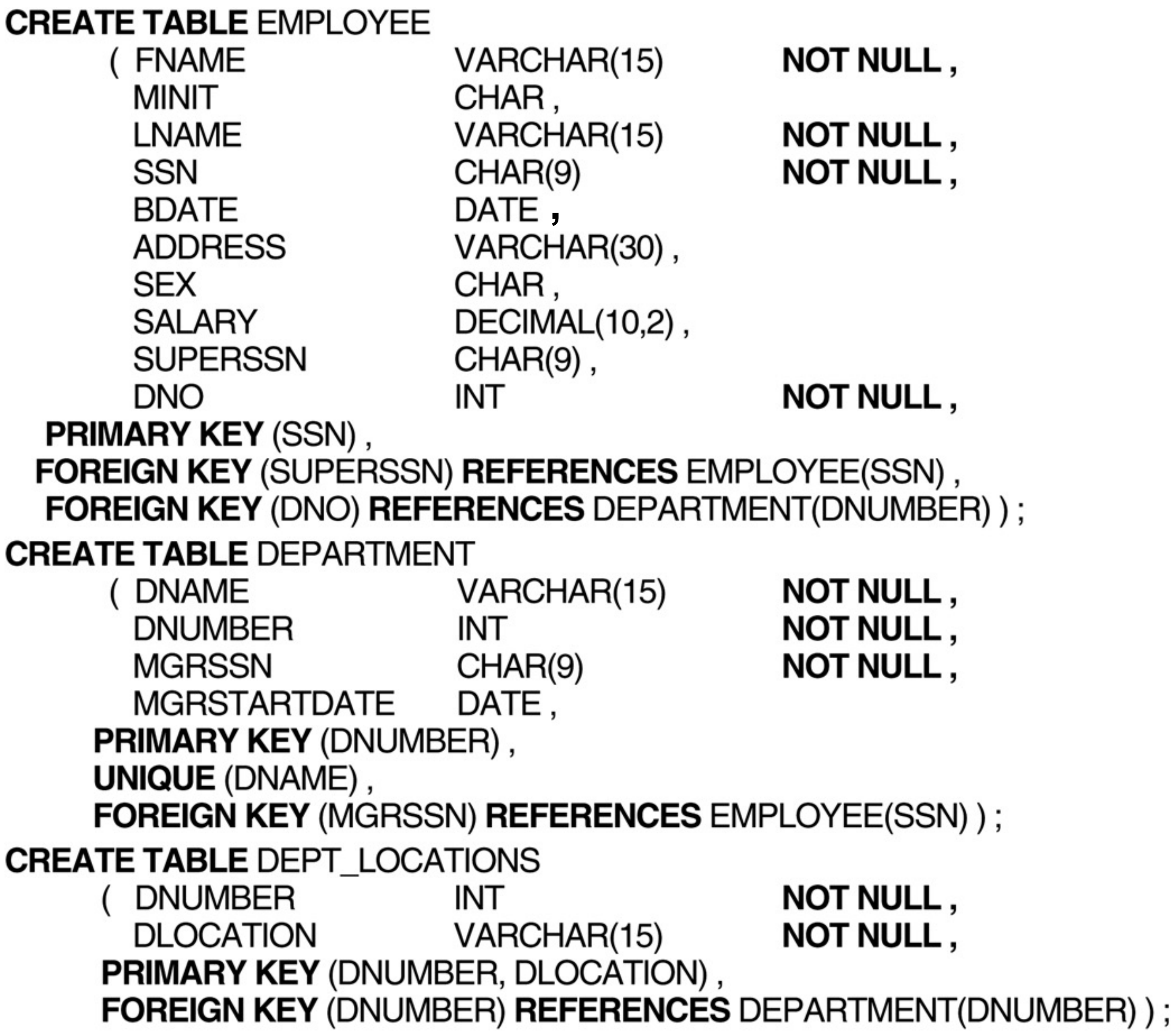
DEPENDENT

ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP

Do create tables & constraints !!

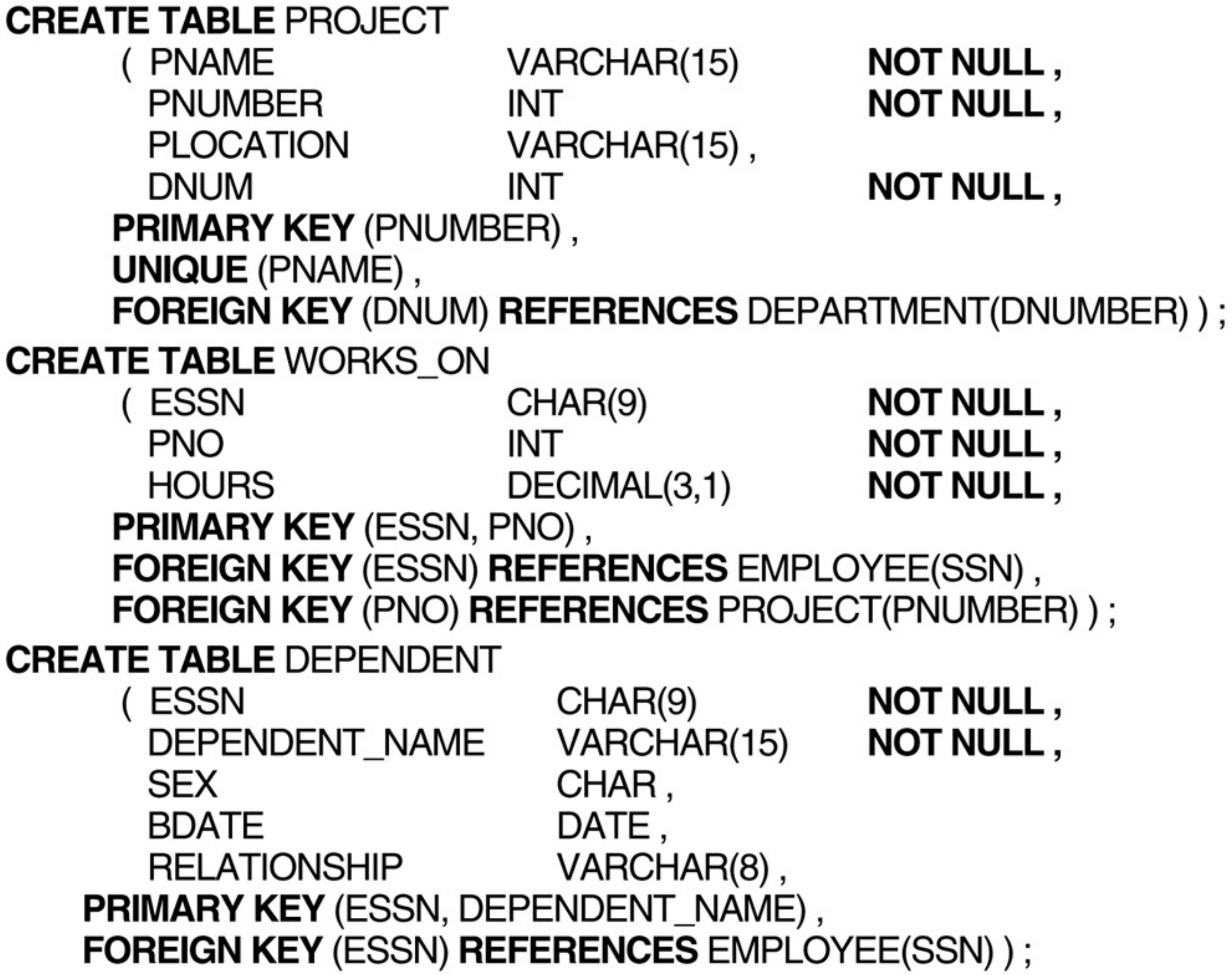
CREATE TABLE TableName
{(colName dataType [NOT NULL]
[UNIQUE]
[DEFAULT defaultOption]
[CHECK searchCondition] [,...]}
[PRIMARY KEY (listOfColumns),]
{[UNIQUE (listOfColumns),] […,]}
{[FOREIGN KEY (listOfFKColumns)
REFERENCES ParentTableName
[(listOfCKColumns)],
[ON UPDATE referentialAction]
[ON DELETE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

Defining the COMPANY DB schema (1)



```
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) ,
          SUPERSSN            CHAR(9) ,
          DNO                 INT                 NOT NULL ,
   PRIMARY KEY (SSN) ,
  FOREIGN KEY (SUPERSSN) REFERENCES EMPLOYEE(SSN) ,
   FOREIGN KEY (DNO) REFERENCES DEPARTMENT(DNUMBER) ) ;
CREATE TABLE DEPARTMENT
        ( DNAME               VARCHAR(15)         NOT NULL ,
          DNUMBER             INT                 NOT NULL ,
          MGRSSN              CHAR(9)             NOT NULL ,
          MGRSTARTDATE        DATE ,
       PRIMARY KEY (DNUMBER) ,
       UNIQUE (DNAME) ,
       FOREIGN KEY (MGRSSN) 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) ) ;
```

Defining the COMPANY DB schema (2)



```
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) ) ;
```

DDL: Create, Alter, Drop

CREATE TABLE

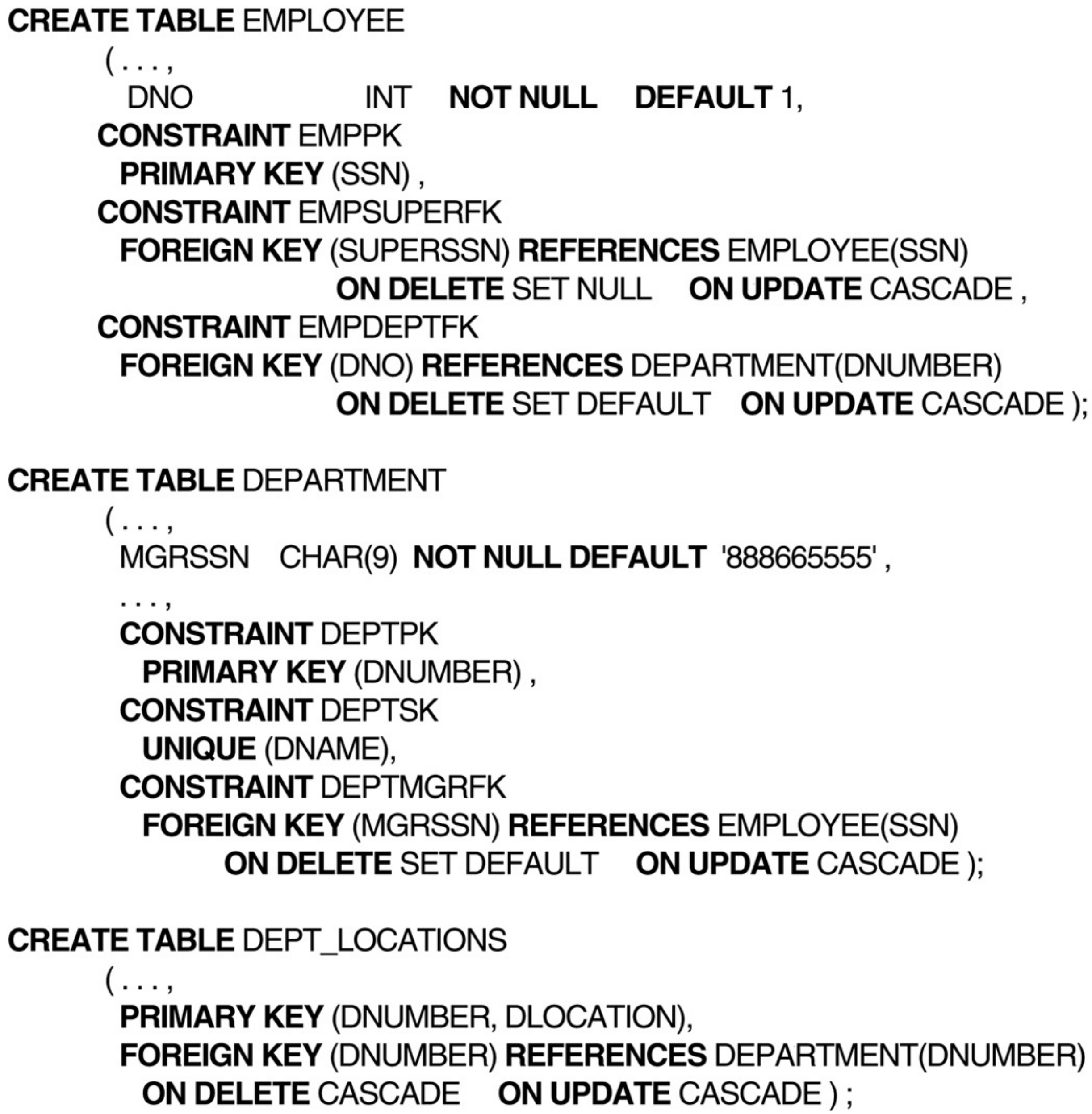
- Default values
  - DEFAULT <value> can be specified for an attribute
  - If no default clause is specified, the default value is NULL for attributes that do not have the NOT NULL constraint

    →If NOT NULL option is specified on attribute A and no value is specified as inserting a tupe r(…A…) ??
  - CHECK clause:

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

  - CREATE DOMAIN can also be used in conjunction with the CHECK clause:

```
CREATE DOMAIN D_NUM AS INTEGER CHECK (D_NUM>0 AND
   D_NUM<21);
```



```
CREATE TABLE EMPLOYEE
        ( . . . ,
          DNO              INT     NOT NULL    DEFAULT 1,
        CONSTRAINT EMPPK
          PRIMARY KEY (SSN) ,
        CONSTRAINT EMPSUPERFK
          FOREIGN KEY (SUPERSSN) 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
        ( . . . ,
          MGRSSN   CHAR(9)  NOT NULL DEFAULT  '888665555' ,
          . . . ,
          CONSTRAINT DEPTPK
            PRIMARY KEY (DNUMBER) ,
          CONSTRAINT DEPTSK
            UNIQUE (DNAME),
          CONSTRAINT DEPTMGRFK
            FOREIGN KEY (MGRSSN) 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 ) ;
```

DDL: Create, Alter, Drop

CREATE TABLE

- Primary key and referential integrity constraints
  - If a PK has a single attribute:

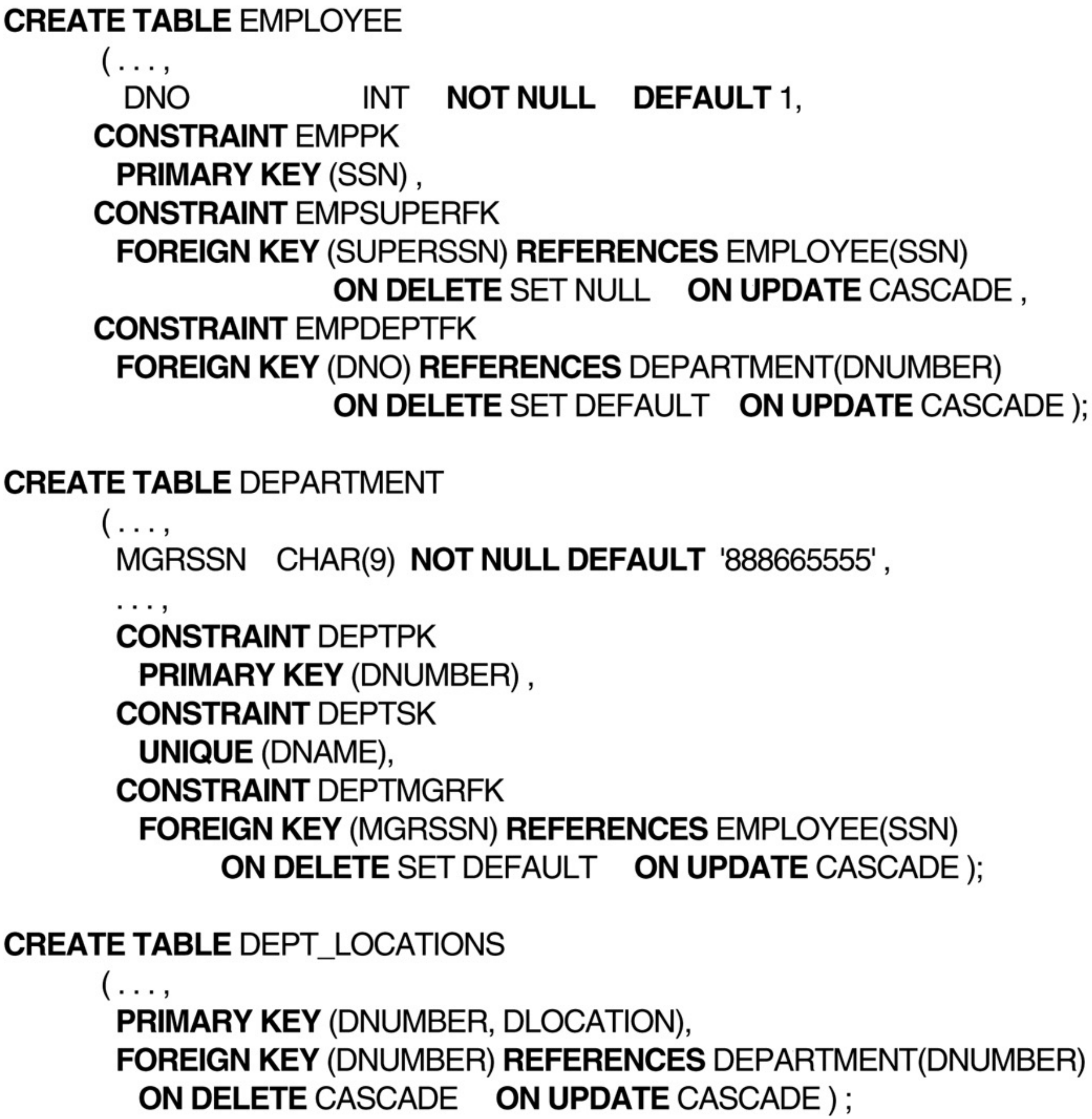
    DNUMBER INT PRIMARY KEY;
  - Referential integrity:

    FOREIGN KEY (list_of_attr) …
  - When are referential integrity constraints violated ?? Default action ??
  - The schema designer can add a **referential triggered action** clause to any FK constraint:

    ON DELETE <action>

    ON UPDATE <action>

    <action>: SET NULL, CASCADE, SET DEFAULT



```
CREATE TABLE EMPLOYEE
        ( . . . ,
         DNO                INT    NOT NULL    DEFAULT 1,
        CONSTRAINT EMPPK
          PRIMARY KEY (SSN) ,
        CONSTRAINT EMPSUPERFK
          FOREIGN KEY (SUPERSSN) 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
        ( . . . ,
          MGRSSN   CHAR(9)  NOT NULL DEFAULT  '888665555' ,
          . . . ,
          CONSTRAINT DEPTPK
            PRIMARY KEY (DNUMBER) ,
          CONSTRAINT DEPTSK
            UNIQUE (DNAME),
          CONSTRAINT DEPTMGRFK
            FOREIGN KEY (MGRSSN) 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 ) ;
```

An example

DDL: Create, Alter, Drop

CREATE TABLE

- Giving names to constraints
  - This is optional
  - 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 (SUPERSSN) 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
        ( . . . ,
          MGRSSN    CHAR(9)  NOT NULL DEFAULT  '888665555' ,
          . . . ,
          CONSTRAINT DEPTPK
            PRIMARY KEY (DNUMBER) ,
          CONSTRAINT DEPTSK
            UNIQUE (DNAME),
          CONSTRAINT DEPTMGRFK
            FOREIGN KEY (MGRSSN) 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 ) ;
```

DDL: Create, Alter, Drop

CREATE TABLE

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

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

  - More general constraints: CREATE ASSERTION

DDL: Create, Alter, Drop

DROP Command

- Used to drop *named* schema elements: tables, domains, constraints, and the schema itself

```
DROP SCHEMA Company CASCADE;
```

or

```
DROP SCHEMA Company RESTRICT;
```

DDL: Create, Alter, Drop

DROP Command

- Drop a table:

  DROP TABLE Dependent CASCADE; (RESTRICT)

  - RESTRICT option: dropped on if it is not referenced in any constraints or views
  - CASCADE option: 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

DDL: Create, Alter, Drop

ALTER Command

- Base tables: adding or dropping a column or constraints, changing a column definition. Example:

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

- Job value for each tuple: default clause or UPDATE command
- What value does each tuple take wrt. the attribute Job if:

```
ALTER TABLE Company.Employee ADD Job VARCHAR(15) NOT
   NULL;
```

- See chapter 6 [1] for the answer & details

DDL: Create, Alter, Drop

ALTER Command

- Drop a column: similarly to drop a table, CASCADE or RESTRICT option must be specified
  - CASCADE option: all constraints and views referencing the column are dropped along with the column
  - RESTRICT option: successful only if no constraints and views are referencing the column

  E.g., ALTER TABLE Company.Employee DROP Address CASCADE;
- Much more details: see [1] & the Web

Outline

- The COMPANY Database
- SQL developments: an overview
- SQL
  - DDL: create, alter, drop
  - DML: select, insert, update, delete
  - DCL: commit, rollback, grant, revoke
- Reading Suggestion:
  - [1]: Chapters 6, 7
  - http://www.oracle.com

DML: Select, Insert, Update, Delete

SELECT

- SQL has one basic statement for retrieving information from a database: the SELECT statement
- This is *not the same as* the SELECT operation of the relational algebra
- Important distinction between SQL and the formal relational model; SQL allows a table (relation) to have two or more tuples that are identical in all their attribute values
- Hence, an 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

DML: Select, Insert, Update, Delete

SELECT

- *Basic form* of the SQL SELECT statement is called a *mapping* or a *SELECT-FROM-WHERE block*

```
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

DML: Select, Insert, Update, Delete

SELECT

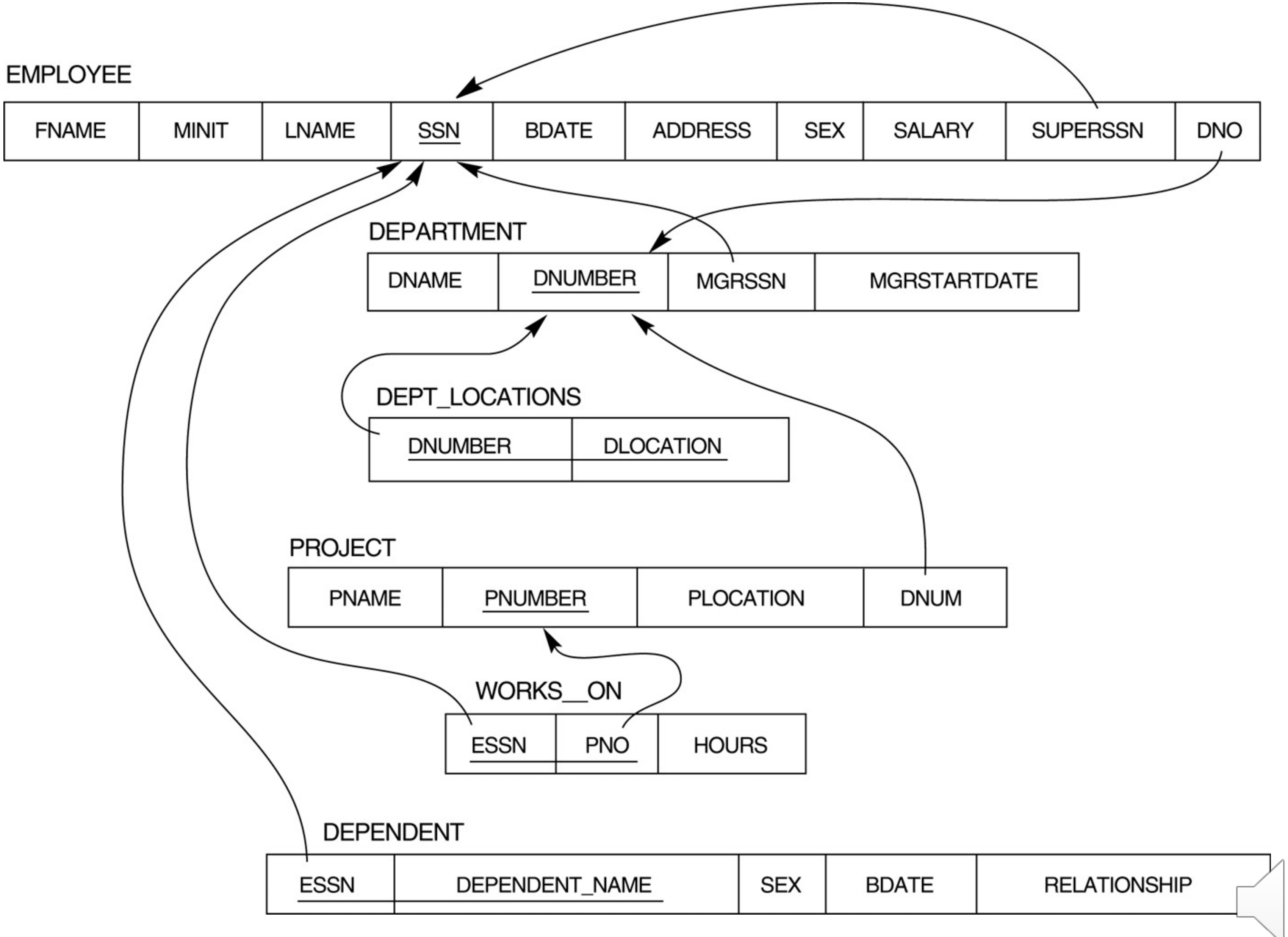
```
SELECT [DISTINCT | ALL]
  {* | [columnExpression [AS newName]]
  [,...] }
FROM TableName [alias] [, ...]
[WHERE condition]
[GROUP BY columnList]  [HAVING
  condition]
[ORDER BY columnList]
```

DML: Select, Insert, Update, Delete

SELECT

- 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



EMPLOYEE

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO

DEPARTMENT

DNAME	DNUMBER	MGRSSN	MGRSTARTDATE

DEPT_LOCATIONS

DNUMBER	DLOCATION

PROJECT

PNAME	PNUMBER	PLOCATION	DNUM

WORKS__ON

ESSN	PNO	HOURS

DEPENDENT

ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP

DML: Select, Insert, Update, Delete

SELECT

- Basic SQL queries correspond to 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; the SELECT-clause specifies the *projection attributes* and the WHERE-clause specifies the *selection condition*
- However, the result of the query *may contain* duplicate tuples

DML: Select, Insert, Update, Delete

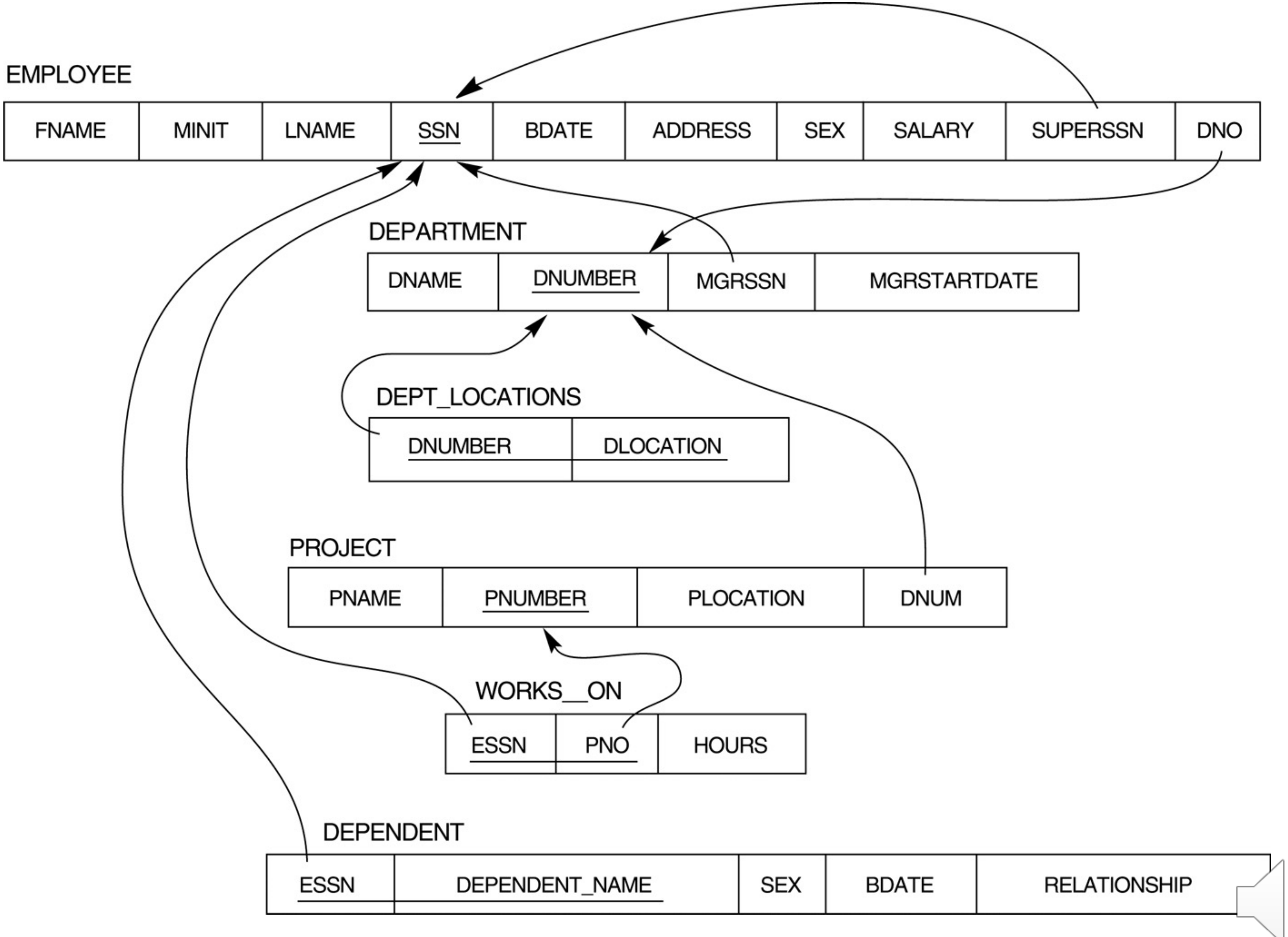
SELECT

- 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
```

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

The COMPANY Database



EMPLOYEE

FNAME | MINIT | LNAME | SSN | BDATE | ADDRESS | SEX | SALARY | SUPERSSN | DNO

DEPARTMENT

DNAME | DNUMBER | MGRSSN | MGRSTARTDATE

DEPT_LOCATIONS

DNUMBER | DLOCATION

PROJECT

PNAME | PNUMBER | PLOCATION | DNUM

WORKS__ON

ESSN | PNO | HOURS

DEPENDENT

ESSN | DEPENDENT_NAME | SEX | BDATE | RELATIONSHIP

DML: Select, Insert, Update, Delete

SELECT

- 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

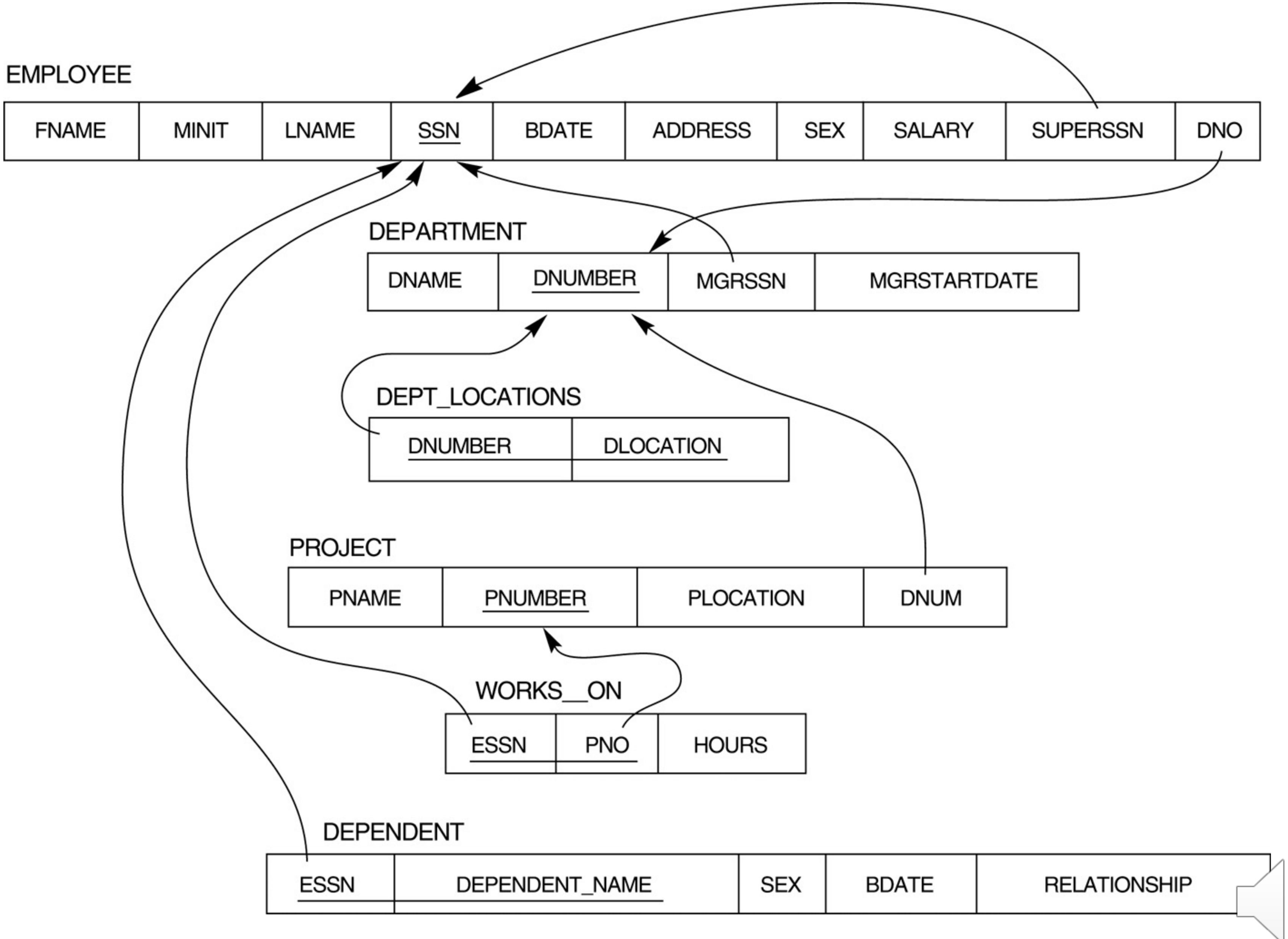
DML: Select, Insert, Update, Delete

SELECT

```
Q2: SELECT      PNUMBER, DNUM, LNAME, BDATE,ADDRESS
    FROM        PROJECT, DEPARTMENT, EMPLOYEE
    WHERE       DNUM=DNUMBER AND MGRSSN=SSN
                AND PLOCATION='Stafford'
```

- There are *2* join conditions:
  - The join condition DNUM=DNUMBER relates a project to its controlling department
  - The join condition MGRSSN=SSN relates the controlling department to the employee who manages that department

The COMPANY Database



EMPLOYEE

FNAME	MINIT	LNAME	<u>SSN</u>	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO

DEPARTMENT

DNAME	<u>DNUMBER</u>	MGRSSN	MGRSTARTDATE

DEPT_LOCATIONS

<u>DNUMBER</u>	<u>DLOCATION</u>

PROJECT

PNAME	<u>PNUMBER</u>	PLOCATION	DNUM

WORKS__ON

<u>ESSN</u>	<u>PNO</u>	HOURS

DEPENDENT

<u>ESSN</u>	<u>DEPENDENT_NAME</u>	SEX	BDATE	RELATIONSHIP

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, 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.

```
Q3:  SELECT         E.FNAME, E.LNAME, S.FNAME, S.LNAME
     FROM           EMPLOYEE E S
     WHERE          E.SUPERSSN=S.SSN
```

  - The alternate relation names E and S are called *aliases* or *tuple variables* for the EMPLOYEE relation
  - We can think of E and S as two *different copies* of EMPLOYEE; E represents employees in role of *supervisees* and 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

```
Q4: 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, MI, LN, SSN, BD, ADDR, SEX, SAL, SSSN,
  DNO)
```

Unspecified WHERE-clause

- A *missing WHERE-clause* indicates no condition; hence, *all tuples* of the relations in the FROM-clause are selected
- This is equivalent to the condition WHERE TRUE
- Query 5: Retrieve the SSN values for all employees

```
Q5:       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
- Example:

```
Q6:  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*
- Examples:

```
Q7: SELECT      *
    FROM        EMPLOYEE
    WHERE       DNO=5

Q8: 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**
- For example, the result of Q9 may have duplicate SALARY values, but Q9A's

```
Q9:     SELECT   SALARY
        FROM     EMPLOYEE

Q9A:    SELECT   DISTINCT SALARY
        FROM     EMPLOYEE
```

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 10: 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 _

```
Q11:     SELECT *
         FROM    Employee
         WHERE Address LIKE ‘%HCMC%’

Q12:     SELECT *
         FROM   Employee
         WHERE BDate LIKE ‘_ _8_ _ _ _ _ _ _’
```

Substring pattern matching and arithmetic operators

- Standard arithmetic operators: +, -, *, /
- Query 13: show the resulting salaries if every employee working on “ProductX” is given 10% raise

```
Q13: SELECT   FNAME, LNAME, 1.1*Salary AS INC_SAL
     FROM     Employee, Works_on, Project
     WHERE    SSN=ESSN AND PNO=PNUMBER AND
              PNAME=‘ProductX’
```

Outline

- The COMPANY Database
- SQL developments: an overview
- SQL
  - DDL: create, alter, drop
  - DML: select, insert, update, delete
  - DCL: commit, rollback, grant, revoke
- Reading Suggestion:
  - [1]: Chapters 6, 7
  - http://www.oracle.com

NULL & 3-valued logic

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

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

NOT	
True	F
False	T
Unknown	U

SELECT * FROM Employee WHERE SuperSSN IS NULL;

SELECT * FROM Employee WHERE SuperSSN IS NOT NULL;

Nested Queries

- A complete SELECT query, called a *nested query* , can be specified within the WHERE-clause of another query, called the *outer query*
- Query 14: Retrieve the name and address of all employees who work for the 'Research' department

```
Q14: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 15: Retrieve the name of each employee who has a dependent with the same first name as the employee.

```
Q15: SELECT   E.FNAME, E.LNAME
       FROM       EMPLOYEE AS E
       WHERE     E.SSN IN (SELECT ESSN
                                    FROM    DEPENDENT
                                    WHERE  ESSN=E.SSN AND
                                      E.FNAME=DEPENDENT_NAME)
```

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, Q15 may be written as in Q15A

```
Q15A:   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 16: 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)

```
Q16: SELECT       DISTINCT       ESSN
       FROM       Works_on
       WHERE      (PNO, HOURS) IN
                  (SELECT        PNO, HOURS
                  FROM           Works_on
                  WHERE          ESSN='123456789')
```

More Comparison Operators

- … {=, >, >=, <, <=, <>} {ANY, SOME, ALL} …
- <u>Query 17:</u> Retrieve all employees whose salary is greater than the salary of all employees in dept. 5

```
Q17: SELECT     *
     FROM       Employee
     WHERE      Salary > ALL (SELECT Salary
                              FROM     Employee
                              WHERE     DNO=5)
```

The EXISTS Function

- EXISTS is used to check if the result of a correlated nested query is empty (contains no tuples)
- Query 15: Retrieve the name of each employee who has a dependent with the same first name as the employee

```
Q15B: SELECT       E.FNAME, E.LNAME
      FROM         EMPLOYEE
      WHERE        EXISTS (SELECT *
                           FROM  DEPENDENT
                           WHERE SSN=ESSN AND
                                 FNAME=DEPENDENT_NAME)
```

The EXISTS Function

- Query 18: Retrieve the names of employees who have no dependents

```
Q18:        SELECT      FNAME, LNAME
            FROM        EMPLOYEE
            WHERE       NOT EXISTS  (SELECT  *
                        FROM  DEPENDENT
                        WHERE SSN=ESSN)
```

- In Q18, the correlated nested query retrieves all DEPENDENT tuples related to an EMPLOYEE tuple. If *none exist* , the EMPLOYEE tuple is selected
- EXISTS is necessary for the expressive power of SQL

Enumerated Sets

- It is also possible to use an **explicit (enumerated) set of values** in the WHERE-clause rather than a nested query
- Query 19: Retrieve the SSNs of all employees who work on project numbers 1, 2, or 3.

```
Q19:    SELECT    DISTINCT ESSN
        FROM      WORKS_ON
        WHERE     PNO IN  (1, 2, 3)
```

Joined Relations Feature in SQL2

- 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)

Joined Relations Feature in SQL2

- Examples:

```
SELECT     E.FNAME, E.LNAME, S.FNAME, S.LNAME
FROM       EMPLOYEE E S
WHERE      E.SUPERSSN=S.SSN
```

can be written as:

```
SELECT     E.FNAME, E.LNAME, S.FNAME, S.LNAME
FROM       (EMPLOYEE E LEFT OUTER JOIN
    EMPLOYEE S ON  E.SUPERSSN=S.SSN)
```

- Any differences ??

Joined Relations Feature in SQL2

- Examples:

```
SELECT      FNAME, LNAME, ADDRESS
FROM        EMPLOYEE, DEPARTMENT
WHERE       DNAME='Research' AND DNUMBER=DNO
```

could be written as:

```
SELECT      FNAME, LNAME, ADDRESS
FROM        (EMPLOYEE JOIN DEPARTMENT ON
            DNUMBER=DNO)
WHERE       DNAME='Research’
```

or as:

```
SELECT      FNAME, LNAME, ADDRESS
FROM        (EMPLOYEE NATURAL JOIN (DEPARTMENT
             AS DEPT(DNAME, DNO, MSSN, MSDATE)))
WHERE       DNAME='Research’
```

Joined Relations Feature in SQL2

- 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 could be written as follows; this illustrates multiple joins in the joined tables

```
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 20: Find the max, min, & average salary among all employees

```
Q20:SELECT        MAX(SALARY), MIN(SALARY), AVG(SALARY)
    FROM          EMPLOYEE
```

AGGREGATE FUNCTIONS

- Queries 21 and 22: Retrieve the total number of employees in the company (Q17), and the number of employees in the 'Research' department (Q18)

```
Q21:SELECT   COUNT (*)
    FROM     EMPLOYEE
Q22: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

- In many cases, we want to 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 function is applied to each subgroup independently
- SQL has a **GROUP BY**-clause for specifying the grouping attributes, which *must also appear in the SELECT-clause*

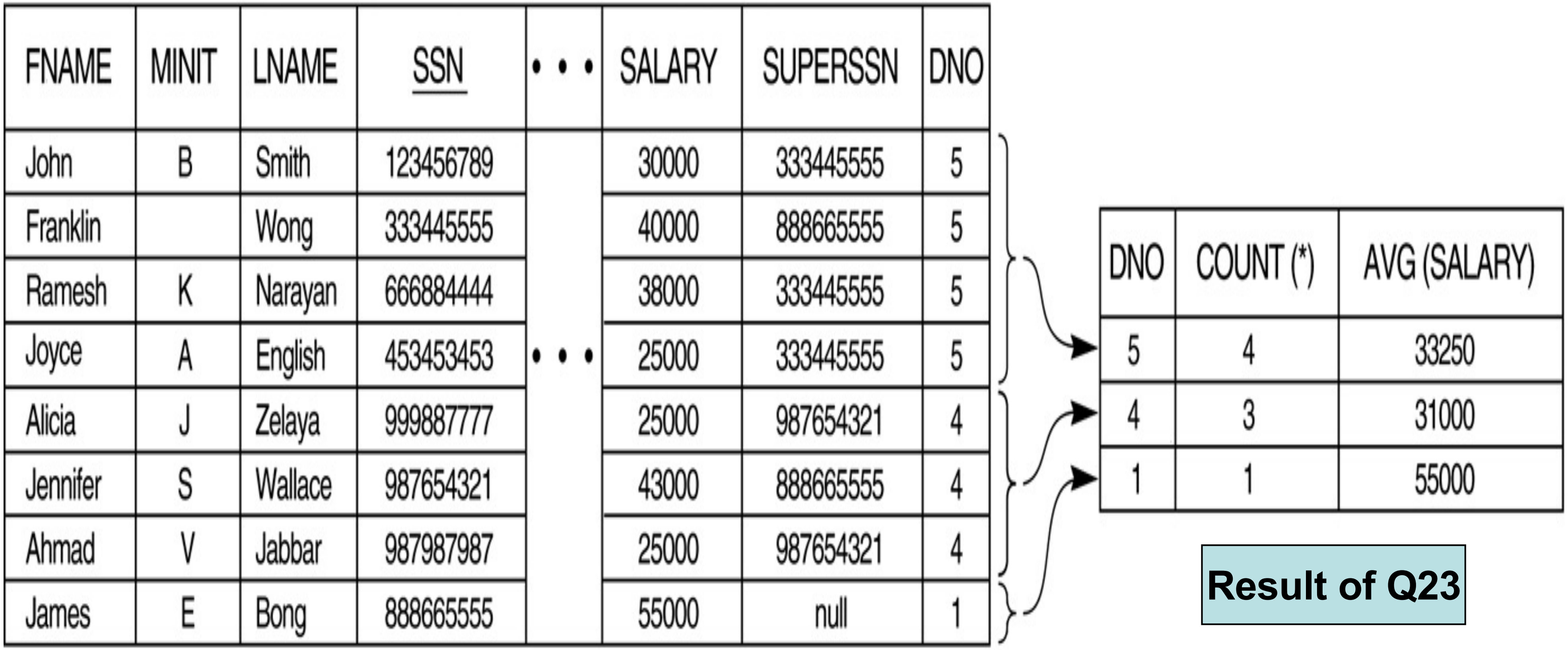
GROUPING

- Query 23: For each department, retrieve the department number, the number of employees in the department, and their average salary

```
Q23: SELECT          DNO, COUNT (*), AVG (SALARY)
     FROM            EMPLOYEE
     GROUP BY        DNO
```

  - In Q23, 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: Q23 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 Q23

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 24: 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.

```
Q24:        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 25: 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

```
Q25: 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 – summarization

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

DML: Select, Insert, Update, Delete

SELECT

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

DML: Select, Insert, Update, Delete

SELECT – Query Optimization

- Chapter 19: homework !!

Outline

- The COMPANY Database
- SQL developments: an overview
- SQL
  - DDL: create, alter, drop
  - DML: select, insert, update, delete
  - DCL: commit, rollback, grant, revoke
- Reading Suggestion:
  - [1]: Chapters 6, 7
  - http://www.oracle.com

DML: Select, Insert, Update, Delete

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

DML: Select, Insert, Update, Delete

Insert

- Example:

```
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')
```

DML: Select, Insert, Update, Delete

Insert

- 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

DML: Select, Insert, Update, Delete

Insert

- 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;
```

DML: Select, Insert, Update, Delete

Delete

- Removes tuples from a relation
- Includes a WHERE-clause to select the tuples to be deleted
- 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

DML: Select, Insert, Update, Delete

Delete

- 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
```

DML: Select, Insert, Update, Delete

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 should be enforced

DML: Select, Insert, Update, Delete

Update

- 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
```

DML: Select, Insert, Update, Delete

Update

- 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

- “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 & as certain conditions are satisfied
- Details of triggers: [1] and Oracle’s website

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

- SQL command: **CREATE VIEW**
  - a view (table) name
  - a possible list of attribute names
  - a query to specify the view contents
- 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
    GROUP       BY PNAME;
```

VIEWs

- 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;
```

- Updating views and concerned issues: HW !!

DCL: Commit, Rollback, Grant, Revoke

- [1] for the details

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)
  - Homework: stored procedures, triggers in Oracle & PL/SQL
- Next lecture:
  - Exercises, presentations
  - Functional dependencies & normalization

Q&A

Question ?