Database Management System (21CS53)

Module-2: Relational Model

Relational Model:

Relational Model Concepts, Relational Model Constraints and relational database schemas, Update operations, transactions, and dealing with constraint violations.

Relational Algebra:

Unary and Binary relational operations, additional relational operations (aggregate, grouping, etc.) Examples of Queries in relational algebra.

Mapping Conceptual Design into a Logical Design:

Relational Database Design using ER-to-Relational mapping.

Textbook 1: Ch 5.1 to 5.3, 8.1 to 8.5, 9.1

THE RELATIONAL DATA MODEL AND RELATIONAL DATABASE CONSTRAINTS

RELATIONAL MODEL CONCEPTS:

Domain: A set/universe of *atomic* values, where by "atomic" mean each value in the domain isindivisible (i.e., cannot be broken down into component parts).

Examples of domains:
USA_phone_number: string of digits of length ten
SSN: string of digits of length nine
Name: string of characters beginning with an upper case letter
GPA: a real number between 0.0 and 4.0
Sex: a member of the set { female, male }
Dept_Code: a member of the set { CMPS, MATH, ENGL, PHYS, PSYC, ... }

These are all *logical* descriptions of domains. For implementation purposes, it is necessary to provide descriptions of domains in terms of concrete **data types** (or **formats**) that are provided by the DBMS (such as String, int, boolean).

Attribute: the *name* of the role played by some value (coming from some domain) in the context of a **relational schema**. The domain of attribute A is denoted dom(A).

Tuple: A tuple is a mapping from attributes to values drawn from the respective domains of those attributes. A tuple is intended to describe some entity (or relationship between entities) in the miniworld.
As an example, a tuple for a PERSON entity might be
{ Name --> "Rumpelstiltskin", Sex --> Male, IQ --> 143 }

Relation: A set of tuples all of the same form (i.e., having the same set of attributes). The term **table** is a loose synonym.

Relational Schema: used for describing the structure of a relation. E.g., R(A1, A2, ..., An) says that R is a relation with *attributes* A1, ... An. The **degree** of a relation is the number of attributes it has, here *n*.

Example: STUDENT(Name, SSN, Address)

Relational Database: A collection of **relations**, each one consistent with its specified relational schema.

CHARACTERISTICS OF RELATIONS

Ordering of Tuples: A relation is a *set* of tuples; hence, there is no order associated with them.

Ordering of Attributes: A tuple is best viewed as a mapping from its attributes. Hence, the order in which the attributes are listed in a table is irrelevant.

Values of Attributes: For a relation to be in *First Normal Form*, each of its attribute domains must consist of atomic (neither composite nor multi-valued) values.
The **Null** value: used for *don't know*, *not applicable*.

Interpretation of a Relation: Each relation can be viewed as a **predicate** and each tuple in that relation can be viewed as an assertion for which that predicate is satisfied (i.e., has value **true**) for the combination of values in it. In other words, each tuple represents a fact. Example: There exists a student having name Benjamin Bayer, having SSN 305-61-2435, having age 19, etc.

Some relations represent facts about entities (e.g., students) whereas others represent facts about relationships (between entities). (e.g., students and course sections).

RELATIONAL MODEL NOTATION:

- $R(A_1, A_2, ..., A_n)$ is a relational schema of degree n denoting that there is a relation R
- having as its attributes $A_1, A_2, ..., A_n$.
- By convention, Q, R, and S denote relation names.
- By convention, q, r, and s denote relation states. For example, $r(R)$ denotes one possible state of relation R. If R is understood from context, this could be written, more simply, as r.
- By convention, t, u, and v denote tuples.

- The "dot notation" *R.A* (e.g., STUDENT.Name) is used to qualify an attribute name, usually for the purpose of distinguishing it from a same-named attribute in a different relation (e.g., DEPARTMENT.Name).

RELATIONAL MODEL CONSTRAINTS

Constraints on databases can be categorized as follows:

- **inherent model-based:** Example: no two tuples in a relation can be duplicates (because a relation is a set of tuples)
- **schema-based:** can be expressed using DDL; this kind is the focus of this section.
- **application-based:** are specific to the "business rules" of the miniworld and typically difficult or impossible to express and enforce within the data model. Hence, it is left to application programs to enforce.

SCHEMA-BASED CONSTRAINTS:

Domain Constraints: Each attribute value must be either **null** (which is really a *non-value*) or drawn from the domain of that attribute. Note that some DBMS's allow you to impose the **not null** constraint upon an attribute, which is to say that that attribute may not have the (non-)value **null**.

Key Constraints: A relation is a *set* of tuples, and each tuple's "identity" is given by the values of its attributes. Hence, it makes no sense for two tuples in a relation to be identical . That is, no two tuples may have the same combination of values in their attributes.

Usually the miniworld dictates that there be subsets of attributes for which no two tuples may have the same combination of values. Such a set of attributes is called a **superkey** of its relation. From the fact that no two tuples can be identical, it follows that the set of all attributes of a relation constitutes a superkey of that relation.

A **key** is a *minimal superkey*, i.e., a superkey such that, if we were to remove any of its attributes, the resulting set of attributes fails to be a superkey.

Example: Suppose that we stipulate that a faculty member is uniquely identified by *Name* and *Address* and also by *Name* and *Department*, but by no single one of the three attributes

mentioned. Then *{ Name, Address, Department }* is a (non-minimal) superkey and each of *{ Name, Address }* and *{ Name, Department }* is a key (i.e., minimal superkey).

Candidate key: any key.

Primary key: a key chosen to act as the means by which to identify tuples in a relation. Typically, one prefers a primary key to be one having as few attributes as possible.

RELATIONAL DATABASES AND RELATIONAL DATABASE SCHEMAS

A **relational database schema** is a set of schemas for its relations together with a set of **integrity constraints**.

A **relational database state/instance/snapshot** is a set of states of its relations such that no integrity constraint is violated.

ENTITY INTEGRITY, REFERENTIAL INTEGRITY, AND FOREIGN KEYS

Entity Integrity Constraint: In a tuple, none of the values of the attributes forming the relation's primary key may have **null value**. Or primary key can not be a null value.

Referential Integrity Constraint: A **foreign key** of relation *R* is a set of its attributes intended to be used for identifying/referring to a tuple in some relation *S*. (*R* is called the *referencing* relation and *S* the *referenced* relation.) For this to make sense, the set of attributes of *R* forming the foreign key should "correspond to" primary key of *S*.

Note that a foreign key may refer to a tuple in the same relation and that a foreign key may be part of a primary key. A foreign key may have value **null.**

SEMANTIC INTEGRITY CONSTRAINTS:

application-specific restrictions that are unlikely to be expressible in DDL. Examples:

- salary of a supervisee cannot be greater than that of her/his supervisor

- salary of an employee cannot be lowered

UPDATE OPERATIONS AND DEALING WITH CONSTRAINT VIOLATIONS.

For each of the *update* operations (Insert, Delete, and Update), we consider what kinds of constraint violations may result from applying it and how we might choose to react.

Insert:

- domain constraint violation: some attribute value is not of correct domain
- entity integrity violation: key of new tuple is **null**
- key constraint violation: key of new tuple is same as existing one
- referential integrity violation: foreign key of new tuple refers to non-existent tuple

Ways of dealing with it: reject the attempt to insert! Or give user opportunity to try again with different attribute values.

Delete:

Referential integrity violation: a tuple referring to the deleted one exists.

Three options for dealing with it:

- Reject the deletion
- Attempt to **cascade** (or **propagate**) by deleting any referencing tuples (plus those that reference them, etc., etc.)
- modify the foreign key attribute values in referencing tuples to **null** or to some valid value referencing a different tuple

Update:

- Key constraint violation: primary key is changed so as to become same as another tuple's
- referential integrity violation:
    - foreign key is changed and new one refers to nonexistent tuple
    - primary key is changed and now other tuples that had referred to this one violate the constraint

TRANSACTIONS:

This concept is relevant in the context where multiple users and/or application programs are accessing and updating the database concurrently. A transaction is a logical unit of work that may involve several accesses and/or updates to the database (such as what might be required to reserve several seats on an airplane flight). The point is that, even though several transactions might be processed concurrently, the end result must be as though the transactions were carried out sequentially. (Example of simultaneous withdrawals from same checking account.)

THE RELATIONAL ALGEBRA

The basic set of operations for the relational model is the **relational algebra**.
These operations enable a user to specify basic retrieval requests as *relational algebra expressions*.
The result of retrieval is a new relation, which may have been formed from one or more relations.

The algebra operations thus produce new relations, which can be further manipulated using operations of the same algebra. A sequence of relational algebra operations forms a **relational algebra expression**, whose result will also be a relation that represents the result of a database query (or retrieval request).

The relational algebra is very important for several reasons.
Provides a formal foundation for relational model operations.
It is used as a basis for implementing and optimizing queries in the query processing and optimization modules that are integral parts of relational database management systems (RDBMSs), Some of its concepts are incorporated into the SQL standard query language for RDBMSs.

RELATIONAL OPERATIONS:

SELECT σ and PROJECT π operations
Set operations: These include UNION, INTERSECTION, DIFFERENCE, CARTESIAN PRODUCT.
JOIN operations ⋈.
Other relational operations: DIVISION, OUTER JOIN, AGGREGATE FUNCTIONS.

SELECT σ and PROJECT π

SELECT operation (denoted by σ):

Selects the tuples (rows) from a relation R that satisfy a certain *selection condition* c
Form of the operation: σc
The condition c is an arbitrary Boolean expression on the attributes of R
Resulting relation has the *same attributes* as R
Resulting relation includes each tuple in r(R) whose attribute values satisfy the condition c

Examples:

$\sigma_{DNO=4}(EMPLOYEE)$

$\sigma_{SALARY>30000}(EMPLOYEE)$

$\sigma_{(DNO=4\ AND\ SALARY>25000)\ OR\ DNO=5}(EMPLOYEE)$

PROJECT operation (denoted by π):

Keeps only certain attributes (columns) from a relation R specified in an *attribute list* L

Form of operation: $\pi_L(R)$

Resulting relation has only those attributes of R specified in L

The PROJECT operation eliminates duplicate tuples in the resulting

Example: $\pi_{SEX,SALARY}(EMPLOYEE)$

If several male employees have salary 30000, only a single tuple <M, 30000> is kept in the resulting relation.

Results of SELECT and PROJECT operations. (a) $\sigma_{(Dno=4\ \textbf{AND}\ Salary>25000)\ OR\ (Dno=5\ \textbf{AND}\ Salary>30000)}(EMPLOYEE)$. (b) $\pi_{Lname,\ Fname,\ Salary}(EMPLOYEE)$. (c) $\pi_{Sex,\ Salary}(EMPLOYEE)$.

(a)

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
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

(b)

Lname	Fname	Salary
Smith	John	30000
Wong	Franklin	40000
Zelaya	Alicia	25000
Wallace	Jennifer	43000
Narayan	Ramesh	38000
English	Joyce	25000
Jabbar	Ahmad	25000
Borg	James	55000

(c)

Sex	Salary
M	30000
M	40000
F	25000
F	43000
M	38000
M	25000
M	55000

Duplicate tuples are eliminated by the π operation.

Sequences of operations: Several operations can be combined to form a *relational algebra expression* (query)

Example: Retrieve the names and salaries of employees who work in department 4:

$\pi_{FNAME,LNAME,SALARY}(\sigma_{DNO=4}(EMPLOYEE))$

Alternatively, we specify explicit intermediate relations for each step:

$$DEPT4\_EMPS \leftarrow \sigma_{DNO=4}(EMPLOYEE)$$

$$\rho_{\leftarrow} \pi_{FNAME,LNAME,SALARY}(DEPT4\_EMPS)$$

Attributes can optionally be *renamed* in the resulting left-hand-side relation (this may be required for some operations that will be presented later):

$$DEPT4\_EMPS \leftarrow \sigma_{DNO=4}(EMPLOYEE)$$

$$\rho_{(FIRSTNAME,LASTNAME,SALARY)} \leftarrow \pi_{FNAME,LNAME,SALARY}(DEPT4\_EMPS)$$

(a)

Fname	Lname	Salary
John	Smith	30000
Franklin	Wong	40000
Ramesh	Narayan	38000
Joyce	English	25000

Results of a sequence of operations. (a) $\pi_{Fname, Lname, Salary}(\sigma_{Dno=5}(EMPLOYEE))$. (b) Using intermediate relations and renaming of attributes.

(b)

TEMP

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
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

R

First_name	Last_name	Salary
John	Smith	30000
Franklin	Wong	40000
Ramesh	Narayan	38000
Joyce	English	25000

RELATIONAL ALGEBRA OPERATION SET THEORY OPERATIONS

UNION: $R1 \cup R2$, **INTERSECTION**: $R1 \cap R2$, **SET DIFFERENCE**: $R1 - R2$,

CARTESIAN PRODUCT: $R1 \times R2$

For U , ∩, -, the operand relations R1(A1, A2, ..., An) and R2(B1, B2, ..., Bn) must have the same number of attributes, and the domains of corresponding attributes must be compatible; that is, dom(Ai) = dom(Bi) for i=1, 2, ..., n. This condition is called union compatibility. Theresulting relation for U , or - has the same attribute names as the first operand relation R1 (by convention).

The set operations UNION, INTERSECTION, and MINUS. (a) Two union-compatible relations. (b) STUDENT ∪ INSTRUCTOR. (c) STUDENT ∩ INSTRUCTOR. (d) STUDENT – INSTRUCTOR. (e) INSTRUCTOR – STUDENT.

(a) **STUDENT**

Fn	Ln
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

INSTRUCTOR

Fname	Lname
John	Smith
Ricardo	Browne
Susan	Yao
Francis	Johnson
Ramesh	Shah

(b)

Fn	Ln
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert
John	Smith
Ricardo	Browne
Francis	Johnson

(c)

Fn	Ln
Susan	Yao
Ramesh	Shah

(d)

Fn	Ln
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

(e)

Fname	Lname
John	Smith
Ricardo	Browne
Francis	Johnson

CARTESIAN PRODUCT

$R(A_1, A_2, ..., A_m, B_1, B_2, ..., B_n) \leftarrow R_1(A_1, A_2, ..., A_m) \times R_2(B_1, B_2, ..., B_n)$

A tuple t exists in R for each combination of tuples t_1 from R_1 and t_2 from R_2 such that: $t[A_1, A_2, ..., A_m] = t_1$ and $t[B_1, B_2, ..., B_n] = t_2$

If R_1 has n_1 tuples and R_2 has n_2 tuples, then R will have $n_1 * n_2$ tuples.

CARTESIAN PRODUCT is a *meaningless operation* on its own. It can *combine related tuples* from two relations *if followed by the appropriate SELECT operation*.

Example: Combine each DEPARTMENT tuple with the EMPLOYEE tuple of the manager.

DEP_EMP ← DEPARTMENT X EMPLOYEE

DEPT_MANAGER $\leftarrow \sigma_{MGRSSN=SSN}$(DEP_EMP)

The Cartesian Product (Cross Product) operation.

FEMALE_EMPS

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
Alicia	J	Zelaya	999887777	1968-07-19	3321Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291Berry, Bellaire, TX	F	43000	888665555	4
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5

EMPNAMES

Fname	Lname	Ssn
Alicia	Zelaya	999887777
Jennifer	Wallace	987654321
Joyce	English	453453453

EMP_DEPENDENTS

Fname	Lname	Ssn	Essn	Dependent_name	Sex	Bdate	...
Alicia	Zelaya	999887777	333445555	Alice	F	1986-04-05	...
Alicia	Zelaya	999887777	333445555	Theodore	M	1983-10-25	...
Alicia	Zelaya	999887777	333445555	Joy	F	1958-05-03	...
Alicia	Zelaya	999887777	987654321	Abner	M	1942-02-28	...
Alicia	Zelaya	999887777	123456789	Michael	M	1988-01-04	...
Alicia	Zelaya	999887777	123456789	Alice	F	1988-12-30	...
Alicia	Zelaya	999887777	123456789	Elizabeth	F	1967-05-05	...
Jennifer	Wallace	987654321	333445555	Alice	F	1986-04-05	...
Jennifer	Wallace	987654321	333445555	Theodore	M	1983-10-25	...
Jennifer	Wallace	987654321	333445555	Joy	F	1958-05-03	...
Jennifer	Wallace	987654321	987654321	Abner	M	1942-02-28	...
Jennifer	Wallace	987654321	123456789	Michael	M	1988-01-04	...
Jennifer	Wallace	987654321	123456789	Alice	F	1988-12-30	...
Jennifer	Wallace	987654321	123456789	Elizabeth	F	1967-05-05	...
Joyce	English	453453453	333445555	Alice	F	1986-04-05	...
Joyce	English	453453453	333445555	Theodore	M	1983-10-25	...
Joyce	English	453453453	333445555	Joy	F	1958-05-03	...
Joyce	English	453453453	987654321	Abner	M	1942-02-28	...
Joyce	English	453453453	123456789	Michael	M	1988-01-04	...
Joyce	English	453453453	123456789	Alice	F	1988-12-30	...
Joyce	English	453453453	123456789	Elizabeth	F	1967-05-05	...

ACTUAL_DEPENDENTS

Fname	Lname	Ssn	Essn	Dependent_name	Sex	Bdate	...
Jennifer	Wallace	987654321	987654321	Abner	M	1942-02-28	...

RESULT

Fname	Lname	Dependent_name
Jennifer	Wallace	Abner

JOIN OPERATIONS

THETA JOIN: Similar to a CARTESIAN PRODUCT followed by a SELECT. The condition c is called a *join condition*.

$R(A_1, A_2, ..., A_m, B_1, B_2, ..., B_n) \leftarrow R_1(A_1, A_2, ..., A_m) \bowtie_c R_2 (B_1, B_2, ..., B_n)$

EQUIJOIN: The join condition c includes one or more *equality comparisons* involving attributes from R_1 and R_2. That is, c is of the form:

$(A_i=B_j)$ AND ... AND $(A_h=B_k)$; $1 \leq i,h \leq m$, $1 \leq j,k \leq n$

In the above EQUIJOIN operation:

$A_i, ..., A_h$ are called the **join attributes** of R_1

$B_j, ..., B_k$ are called the **join attributes** of R_2

Example of using EQUIJOIN:

Retrieve each DEPARTMENT's name and its manager's name:

$T \leftarrow DEPARTMENT \bowtie_{MGRSSN = SSN} EMPLOYEE$

$RESULT \leftarrow \pi_{DNAME,FNAME,LNAME} (T)$

NATURAL JOIN (*):

In an EQUIJOIN $R \leftarrow R_1 \bowtie_c R_2$, the join attribute of R_2 appear *redundantly* in the result relation R. In a NATURAL JOIN, the *redundant join attributes* of R_2 are *eliminated* from R.

The equality condition is *implied* and need not be specified.

R ←R1 * (join attributes of R1), (join attributes of R2) R2

Example: Retrieve each EMPLOYEE's name and the name of the DEPARTMENT he/she works for:

$T \leftarrow EMPLOYEE *_{(DNO), (DNUMBER)} DEPARTMENT$

$RESULT \leftarrow \pi_{FNAME,LNAME,DNAME} (T)$

If the join attributes *have the same names* in both relations, they *need not be specified* and we can write R ←R1 * R2.

Example: Retrieve each EMPLOYEE's name and the name of his/her SUPERVISOR:

SUPERVISOR(SUPERSSN,SFN,SLN) ←π SSN,FNAME,LNAM (EMPLOYEE)

T← EMPLOYEE * SUPERVISOR

RESULT ← π FNAME,LNAME,SFN,SLN (T)

Note: In the *original definition* of NATURAL JOIN, the join attributes were *required* to have the same names in both relations.

There can be a more than one set of join attributes with a different meaning between the same.

JOIN ATTRIBUTES	RELATIONSHIP
EMPLOYEE.SSN=DEPARTMENT.MGRSSN	EMPLOYEE manages the DEPARTMENT
EMPLOYEE.DNO=DEPARTMRNT.DNUMBER	EMPLOYEE works for the DEPARTMENT

(a)

PROJ_DEPT

Pname	Pnumber	Plocation	Dnum	Dname	Mgr_ssn	Mgr_start_date
ProductX	1	Bellaire	5	Research	333445555	1988-05-22
ProductY	2	Sugarland	5	Research	333445555	1988-05-22
ProductZ	3	Houston	5	Research	333445555	1988-05-22
Computerization	10	Stafford	4	Administration	987654321	1995-01-01
Reorganization	20	Houston	1	Headquarters	888665555	1981-06-19
Newbenefits	30	Stafford	4	Administration	987654321	1995-01-01

(b)

DEPT_LOCS

Dname	Dnumber	Mgr_ssn	Mgr_start_date	Location
Headquarters	1	888665555	1981-06-19	Houston
Administration	4	987654321	1995-01-01	Stafford
Research	5	333445555	1988-05-22	Bellaire
Research	5	333445555	1988-05-22	Sugarland
Research	5	333445555	1988-05-22	Houston

Results of two NATURAL JOIN operations. (a) PROJ_DEPT ← PROJECT * DEPT.
(b) DEPT_LOCS ← DEPARTMENT * DEPT_LOCATIONS.

Example: Retrieve each EMPLOYEE's name and the name of the DEPARTMENT he/she works for:

$T \leftarrow EMPLOYEE \bowtie_{DNO=DNUMBER} DEPARTMENT$

$RESULT \leftarrow \pi_{FNAME,LNAME,DNAME} (T)$

A relation can have a *set of join attributes* to join it with *itself* :

JOIN ATTRIBUTES	RELATIONSHIP
EMPLOYEE(1).SUPERSSN=	EMPLOYEE(2) *supervises*
EMPLOYEE(2).SSN	EMPLOYEE(1)

One can *think of this* as joining *two distinct copies* of the relation, although only one relation actually exists In this case, *renaming* can be useful.

Example: Retrieve each EMPLOYEE's name and the name of his/her SUPERVISOR:

$SUPERVISOR(SSSN,SFN,SLN) \leftarrow \pi_{SSN,FNAME,LNAME}(EMPLOYEE)$

$T \leftarrow EMPLOYEE \bowtie_{SUPERSSN=SSSN} SUPERVISOR$

$RESULT \leftarrow \pi_{FNAME,LNAME,SFN,SLN} (T)$

DIVISION OPERATION

The DIVISION operation, denoted by ÷, is useful for a special kind of query that sometimes occurs in database applications.

An example is *Retrieve the names of employees who work on **all** the projects that 'John Smith' works on.*

First, retrieve the list of project numbers that 'John Smith' works on in the intermediate relation

SMITH_PNOS:

SMITH ← σFname='John' **AND** Lname='Smith'(EMPLOYEE)

SMITH_PNOS ← πPno(WORKS_ON ⋈ Essn=SsnSMITH)

Next, create a relation that includes a tuple <Pno, Essn> whenever the employee whose Ssn is Essn works on the project whose number is Pno in the intermediate relation SSN_PNOS:

SSN_PNOS ← πEssn, Pno(WORKS_ON)

Finally, apply the DIVISION operation to the two relations, which gives the desired employees' Social Security numbers:

SSNS(Ssn) ← SSN_PNOS ÷ SMITH_PNOS

RESULT ← πFname, Lname(SSNS * EMPLOYEE)

The DIVISION operation. (a) Dividing SSN_PNOS by SMITH_PNOS. (b) $T \leftarrow R \div S$.

(a)

SSN_PNOS

Essn	Pno
123456789	1
123456789	2
666884444	3
453453453	1
453453453	2
333445555	2
333445555	3
333445555	10
333445555	20
999887777	30
999887777	10
987987987	10
987987987	30
987654321	30
987654321	20
888665555	20

SMITH_PNOS

Pno
1
2

SSNS

Ssn
123456789
453453453

(b)

R

A	B
a1	b1
a2	b1
a3	b1
a4	b1
a1	b2
a3	b2
a2	b3
a3	b3
a4	b3
a1	b4
a2	b4
a3	b4

S

A
a1
a2
a3

T

B
b1
b4

AGGREGATE FUNCTIONS (ℑ)

Functions such as SUM, COUNT, AVERAGE, MIN, MAX are often applied to sets of values or sets of tuples in database applications

<grouping attributes> ℑ<function list>(R)

The grouping attributes are optional

Example 1: Retrieve the average salary of all employees (no grouping needed):

ρ(AVGSAL)←ℑ AVERAGE SALARY (EMPLOYEE)

Example 2: For each department, retrieve the department number, the number of employees, and the average salary (in the department):

ρ (DNO,NUMEMPS,AVGSAL) ← DNO ℑ COUNT SSN, AVERAGE SALARY (EMPLOYEE)

DNO is called the *grouping attribute* in the above example

The aggregate function operation.

a. $\rho_{R(Dno,\ No\_of\_employees,\ Average\_sal)}({}_{Dno}\Im_{COUNT\ Ssn,\ AVERAGE\ Salary}(EMPLOYEE))$.

b. ${}_{Dno}\Im_{COUNT\ Ssn,\ AVERAGE\ Salary}(EMPLOYEE)$.

c. $\Im_{COUNT\ Ssn,\ AVERAGE\ Salary}(EMPLOYEE)$.

R

(a)

Dno	No_of_employees	Average_sal
5	4	33250
4	3	31000
1	1	55000

(b)

Dno	Count_ssn	Average_salary
5	4	33250
4	3	31000
1	1	55000

(c)

Count_ssn	Average_salary
8	35125

OUTER JOIN

In a regular EQUIJOIN or NATURAL JOIN operation, tuples in R_1 or R_2 that do not have matching tuples in the other relation *do not appear in the result.*

Some questions require all tuples in R1 (or R2 or both) to appear in the result.

When no matching tuples are found, nulls are placed for the missing attributes.

LEFT OUTER JOIN(⟕): lets every tuple in R1 appear in the result

Employee

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales
Tim	1123	Executive

Dept

DeptName	Manager
Sales	Harriet
Production	Charles

Employee ⟕ Dept

Name	EmpId	DeptName	Manager
Harry	3415	Finance	
Sally	2241	Sales	Harriet
George	3401	Finance	
Harriet	2202	Sales	Harriet
Tim	1123	Executive	

RIGHT OUTER JOIN(⟖): lets every tuple in R2 appear in the result.

Employee

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales
Tim	1123	Executive

Dept

DeptName	Manager
Sales	Harriet
Production	Charles

Employee ⟖ Dept

Name	EmpId	DeptName	Manager
Sally	2241	Sales	Harriet
Harriet	2202	Sales	Harriet
		Production	Charles

FULL OUTER JOIN (⟗): lets evey tuple in R1 or R2 appear in the result

Employee

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales
Tim	1123	Executive

Dept

DeptName	Manager
Sales	Harriet
Production	Charles

Employee ⟗ Dept

Name	EmpId	DeptName	Manager
Harry	3415	Finance	
Sally	2241	Sales	Harriet
George	3401	Finance	
Harriet	2202	Sales	Harriet
Tim	1123	Executive	
		Production	Charles

Relational Database Design Using ER-to-Relational Mapping

ER-to-Relational Mapping Algorithm

Step 1: Mapping of Regular Entity Types. For each regular (strong) entity type *E* in the ER schema, create a relation *R* that includes all the simple attributes of *E*. Include only the simple component attributes of a composite attribute. Choose one of the key attributes of *E* as the primary key for *R*. If the chosen key of *E* is a composite, then the set of simple attributes that form it will together form the primary key of *R*.

Step 2: Mapping of Weak Entity Types. For each weak entity type *W* in the ER schema with owner entity type *E*, create a relation *R* and include all simple attributes (or simple components of composite attributes) of *W* as attributes of *R*. In addition, include as foreign key attributes of *R*, the primary key attribute(s) of the relation(s) that correspond to the ownerentity type(s); this takes care of mapping the identifying relationship type of *W*. The primary key of *R* is the combination of the primary key(s) of the owner(s) and the partial key of the weak entity type *W*, if any.

If there is a weak entity type *E*2 whose owner is also a weak entity type *E*1, then *E*1 should be mapped before *E*2 to determine its primary key first.

Step 3: Mapping of Binary 1:1 Relationship Types. For each binary 1:1 relationship type *R* in the ER schema, identify the relations *S* and *T* that correspond to the entity types participating in *R*. There are three possible approaches: (1) the foreign key approach, (2) the merged relationship approach, and (3) the cross reference or relationship relation approach. The first approach is the most useful and should be followed unless special conditions exist,as we discuss below.

1. **Foreign key approach:** Choose one of the relations—*S*, say—and include as a foreign key in *S* the primary key of *T*. It is better to choose an entity type with *total participation* in *R* in the role of *S*. Include all the simple attributes (or simple components of composite attributes) of the 1:1 relationship type *R* as attributes of *S*.

Note that it is possible to include the primary key of *S* as a foreign key in *T* instead.

2. **Merged relation approach:** An alternative mapping of a 1:1 relationship type is to merge the two entity types and the relationship into a single relation. This is possible when *both participations are total*, as this would indicate that the two tables will have the exact same number of tuples at all times.

3. **Cross-reference or relationship relation approach:** The third option is to set up a third relation *R* for the purpose of cross-referencing the primary keys of the two relations *S* and *T* representing the entity types. The relation *R* is called a **relationship relation** (or sometimes a **lookup table**), because each tuple in *R* represents a relationship instance that relates one tuple from *S* with one tuple from *T*. The relation *R* will include the primary key attributes of *S* and *T* as foreign keys to *S* and *T*. The primary key of *R* will be one of the two foreign keys, and the other foreign key will be a unique key of *R*. The drawback is having an extra relation, and requiring an extra join operation when combining related tuples from the tables.

Step 4: Mapping of Binary 1:N Relationship Types. For each regular binary 1:N relationship type *R*, identify the relation *S* that represents the participating entity type at the *N-side* of the relationship type. Include as foreign key in *S* the primary key of the relation *T* that represents the other entity type participating in *R*. Include any simple attributes (or simple components of composite attributes) of the 1:N relationship type as attributes of *S*.

An alternative approach is to use the **relationship relation** (cross-reference) option as in the third option for binary 1:1 relationships. Create a separate relation *R* whose attributes are the primary keys of *S* and *T*, which will also be foreign keys to *S* and *T*. The primary key of *R* is the same as the primary key of *S*. This option can be used if few tuples in *S* participate in the relationship to avoid excessive NULL values in the foreign key.

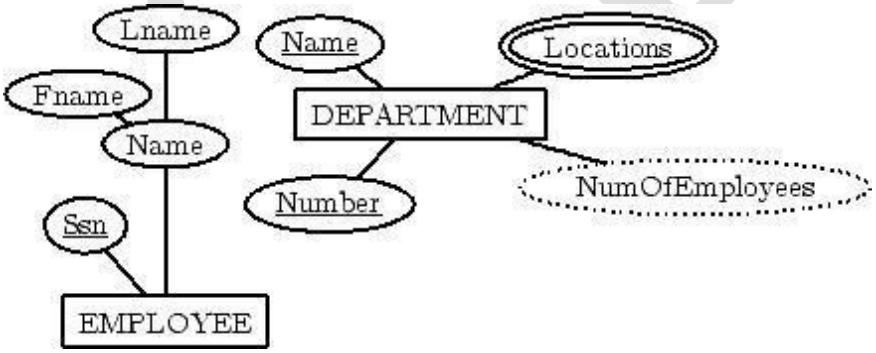
Step 5: Mapping of Binary M:N Relationship Types. For each binary M:N relationship type *R*, create a new relation *S* to represent *R*. Include as foreign key attributes in *S* the primary keys of the relations that represent the participating entity types; their *combination* will form the primary key of *S*. Also include any simple attributes of the M:N relationship type (or simple components of composite attributes) as attributes of *S*.

Step 6: Mapping of Multivalued Attributes. For each multivalued attribute *A*, create a new relation *R*. This relation *R* will include an attribute corresponding to *A*, plus the primary key attribute *K*—as a foreign key in *R*—of the relation that represents the entity type or relationship type that has *A* as a multivalued attribute. The primary key of *R* is the combination of *A* and *K*. If the multivalued attribute is composite, we include its simple components.

Step 7: Mapping of *N*-ary Relationship Types. For each *n*-ary relationship type *R*, where *n* > 2, create a new relation *S* to represent *R*. Include as foreign key attributes in *S* the primary keys of the relations that represent the participating entity types. Also include any simple attributes of the *n*-ary relationship type (or simple components of composite attributes) as attributes of *S*. The primary key of *S* is usually a combination of all the foreign keys that reference the relations representing the participating entity types. However, if the cardinality constraints on any of the entity types *E* participating in *R* is 1, then the primary key of *S* should not include the foreign key attribute that references the relation *E*_ corresponding to*E*.

Example:

Step 1: For each **regular (strong) entity type** E in the ER schema, create a relation R that includes all the simple attributes of E.



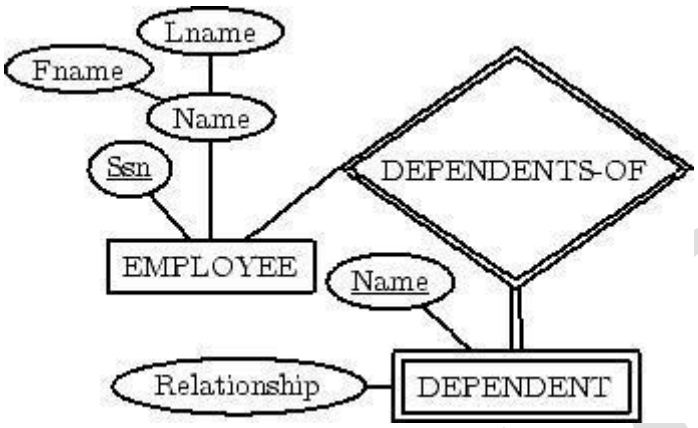
EMPLOYEE

SSN	Fname	Lname

DEPARTMENT

Number	Name

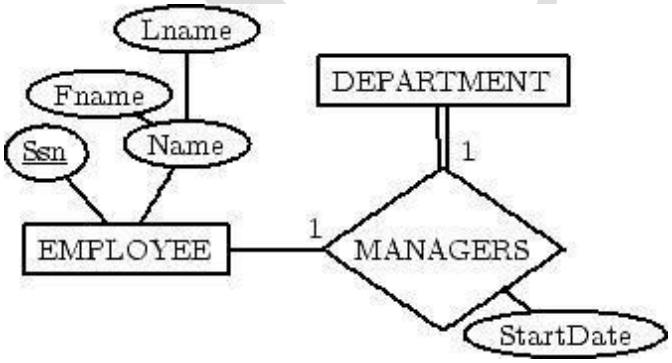
Step 2: For each **weak entity type** W in the ER schema with owner entity type E, create a relation R, and include all simple attributes (or simple components of composite attributes) of W as attributes. In addition, include as foreign key attributes of R the primary key attribute(s) of the relation(s) that correspond to the owner entity type(s).



DEPENDENT

ESSN	Name	Relationship

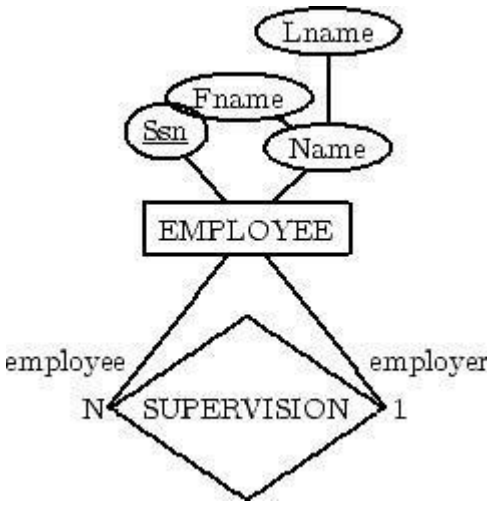
Step 3: For each **binary 1:1 relationship type** R in the ER schema, identify the relations S and T that correspond to the entity types participating in R. Choose one of the relations, sayS, and include the primary key of T as a foreign ke y in S. Include all the simple attributes of R as attributes of S.



MGR_SSN and StartDate should be included in DEPARTMENT

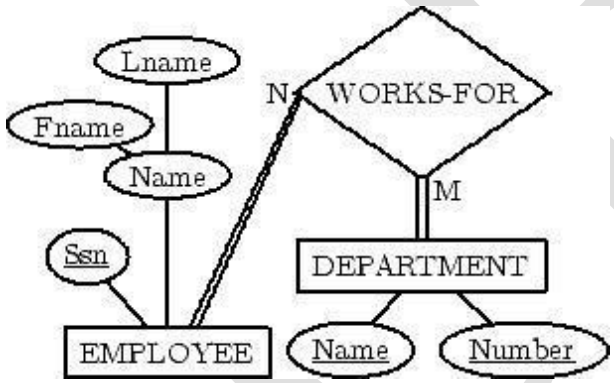
Step 4: For each regular **binary 1:N relationship type** R identify the relation (N) relation S.

Include the primary key of T as a foreign key of S. Simple attributes of R map to attributes of S.



Include SuperSSN in EMPLOYEE.

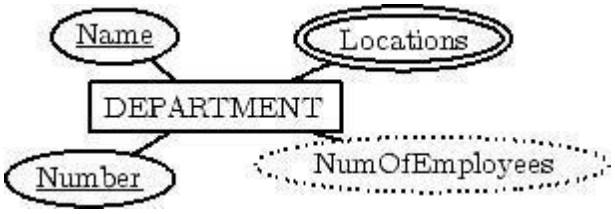
Step 5: For each **binary M:N relationship type** R, create a relation S. Include the primary keys of participant relations as foreign keys in S. Their combination will be the primary key for S. Simple attributes of R become attributes of S.



WORKS-FOR

ESSN	Dnumber

Step 6: For each **multi-valued attribute A**, create a new relation R. This relation will include an attribute corresponding to A, plus the primary key K of the parent relation (entity type or relationship type) as a foreign key in R. The primary key of R is the combination of Aand K.



DEP-LOCATION

Dnumber	Location

Step 7: For each **n-ary relationship type R**, where n>2, create a new relation S to represent R. Include the primary keys of the relations participating in R as foreign keys in S. Simple attributes of R map to attributes of S. The primary key of S is a combination of all the foreign keys that reference the participants that have cardinality constraint >