**MODULE 2**

**The Relational Data Model and Relational Database Constraints and Relational Algebra**

**4.1 Relational Model Concepts**

* **Domain**: A (usually named) set/universe of*atomic*values, where by "atomic" we meansimply that, from the point of view of the database, each value in the domain is indivisible (i.e., cannot be broken down into component parts).

Examples of domains (some taken from page 147):

1. USA\_phone\_number: string of digits of length ten

o SSN: string of digits of length nine

o Name: string of characters beginning with an upper case letter

o GPA: a real number between 0.0 and 4.0

o Sex: a member of the set { female, male }

o 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 asString,int,boolean), in a manneranalogous to how programming languages have intrinsic data types.

* **Attribute**: the*name*of the role played by some value (coming from some domain) in thecontext 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 domainsof 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 (named) set of tuples all of the same form (i.e., having the same set ofattributes). The term **table** is a loose synonym. (Some database purists would argue that a table is "only" a physical manifestation of a relation.)
* **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)

(See Figure 5.1, page 149, for an example of a STUDENT relation/table having several tuples/rows.)

One would think that a "complete" relational schema would also specify the domain of each attribute.

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

**2.1.2 Characteristics of Relations**

**Ordering of Tuples**: A relation is a*set*of tuples; hence, there is no order associated with them.That is, it makes no sense to refer to, for example, the 5th tuple in a relation. When a relation is depicted as a table, the tuples are necessarily listed in *some* order, of course, but you should attach no significance to that order. Similarly, when tuples are represented on a storage device, they must be organized in *some* fashion, and it may be advantageous, from a performance standpoint, to organize them in a way that depends upon their content.

**Ordering of Attributes**: A tuple is best viewed as a mapping from its attributes (i.e., the nameswe give to the roles played by the values comprising the tuple) to the corresponding values. Hence, the order in which the attributes are listed in a table is irrelevant. (Note that, unfortunately, the set theoretic operations in relational algebra (at least how E&N define them) make implicit use of the order of the attributes. Hence, E&N view attributes as being arranged as a sequence rather than a set.)

**Values of Attributes**: For a relation to be in*First Normal Form*, each of its attribute domainsmust consist of atomic (neither composite nor multi-valued) values. Much of the theory underlying the relational model was based upon this assumption. Chapter 10 addresses the issue of including non-atomic values in domains. (Note that in the latest edition of C.J. Date's book, he explicitly argues against this idea, admitting that he has been mistaken in the past.)

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 thatrelation 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 (see Figure 5.1): The first tuple listed means: There exists a student having name Benjamin Bayer, having SSN 305-61-2435, having age 19, etc.

Keep in mind that some relations represent facts about entities (e.g., students) whereas others represent facts about relationships (between entities). (e.g., students and course sections).

The **closed world assumption** states that the only true facts about the miniworld are those represented by whatever tuples currently populate the database.

**2.1.3 Relational Model Notation**: page 152

* *R(A1, A2, ..., An)* is a relational schema of degree *n* denoting that there is a relation *R* having as its attributes *A1, A2, ..., An*.
* 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).





**2.2 Relational Model Constraints and Relational Database Schemas**

Constraints on databases can be categorized as follows:

* **inherent model-based:** Example: no two tuples in a relation can be duplicates (because arelation 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 typicallydifficult or impossible to express and enforce within the data model. Hence, it is left to application programs to enforce.

Elaborating upon **schema-based constraints**:

**2.2.1 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**.

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

Usually the miniworld dictates that there be (proper) 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 attributesmentioned. 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! (Hence, it is not clear what distinguishes a key from a candidate 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.

**2.2.3 Relational Databases and Relational Database Schemas**

A **relational database schema** is a set of schemas for its relations (see Figure 5.5, page 157) 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. (See Figure 5.6, page 159, for a snapshot of COMPANY.)

**2.2.4 Entity Integrity, Referential Integrity, and Foreign Keys**

**Entity Integrity Constraint**: In a tuple, none of the values of the attributes forming therelation's primary key may have the (non-)value **null**. Or is it that at least one such attribute must have a non-null value? In my opinion, E&N do not make it clear!

**Referential Integrity Constraint**: (See Figure 5.7) A **foreign key** of relation*R*is a set of itsattributes intended to be used (by each tuple in *R*) 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" some superkey of *S*. Indeed, by definition we require this superkey to be the primary key of *S*.

This constraint says that, for every tuple in *R*, the tuple in *S* to which it refers must actually be in *S.* Note that a foreign key may refer to a tuple in the same relation and that a foreign key may bepart of a primary key (indeed, for weak entity types, this will always occur). A foreign key may have value **null** (necessarily in all its attributes??), in which case it does not refer to any tuple in the referenced relation.

**Semantic Integrity Constraints**: application-specific restrictions that are unlikely to beexpressible in DDL. Examples:

* salary of a supervisee cannot be greater than that of her/his supervisor
* salary of an employee cannot be lowered

**2.3 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.

**2.3.1 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.

**2.3.2 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

**2.3.3 Update**:

* Key constraint violation: primary key is changed so as to become same as another tuple's
* referential integrity violation:

1. foreign key is changed and new one refers to nonexistent tuple
2. primary key is changed and now other tuples that had referred to this one violate the constraint

**2.3.4 Transactions**: This concept is relevant in the context where multiple users and/orapplication 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**



Operations

to

manipulate relations.



Used

to

specify

retrieval requests (queries).



Query result is in

the form of a relation

**2.4 Relational Operations:**

SELECT

and PROJECT 

operations.

Set operations: These include UNION U, INTERSECTION | |, DIFFERENCE -, CARTESIAN PRODUCT X.

JOIN operations  .

Other relational operations: DIVISION, OUTER JOIN, AGGREGATE FUNCTIONS.

**2.4.1 SELECT  and PROJECT **

**SELECT** operation (denoted by  ):



Selects the tuples (rows) from a relation R that satisfy a certain

*selection condition* c





Form of the operation: c

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:

DNO=4(EMPLOYEE)

SALARY>30000(EMPLOYEE)



(DNO=4 AND SALARY>25000) OR DNO=5 (EMPLOYEE)

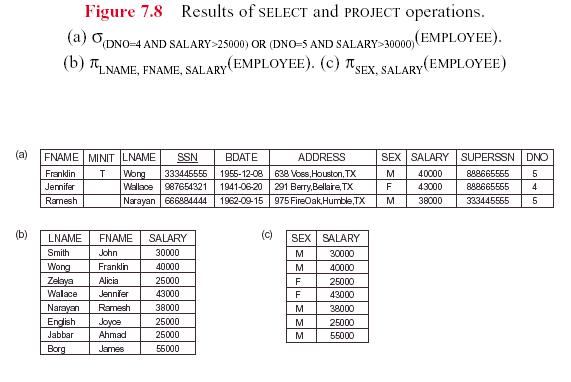
**PROJECT** operation (denoted by ):

|  |  |  |
| --- | --- | --- |
|  | Keeps only certain | |
| attributes (columns) from a relation R specified in an *attribute list* L | |  |
|  | Form | of |
| operation: | L(R) |  |
|  | Resulting | relation |
| has only those attributes of R specified in L | |  |
|  | The PROJECT operation eliminates duplicate tuples in the resulting | |
| relation so that it remains a mathematical set (no duplicate elements). | |  |



Example: SEX,SALARY(EMPLOYEE)

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



**Duplicate tuples are eliminated by the  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:



FNAME,LNAME,SALARY ( DNO=4(EMPLOYEE) )

*Alternatively*, we specify explicit intermediate relations for each

step:

DEPT4\_EMPS DNO=4(EMPLOYEE)

 (DEPT4\_EMPS)

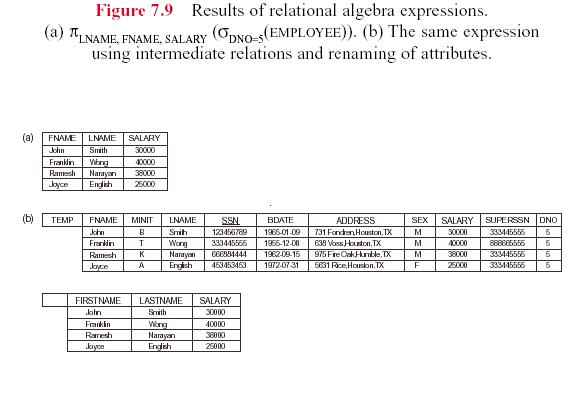


FNAME,LNAME,SALARY

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 DNO=4(EMPLOYEE)

 (FIRSTNAME,LASTNAME,SALARY)  FNAME,LNAME,SALARY(DEPT4\_EMPS)



**2.5 Relational algebra operation Set theory Operations**

Binary operations from mathematical set theory:

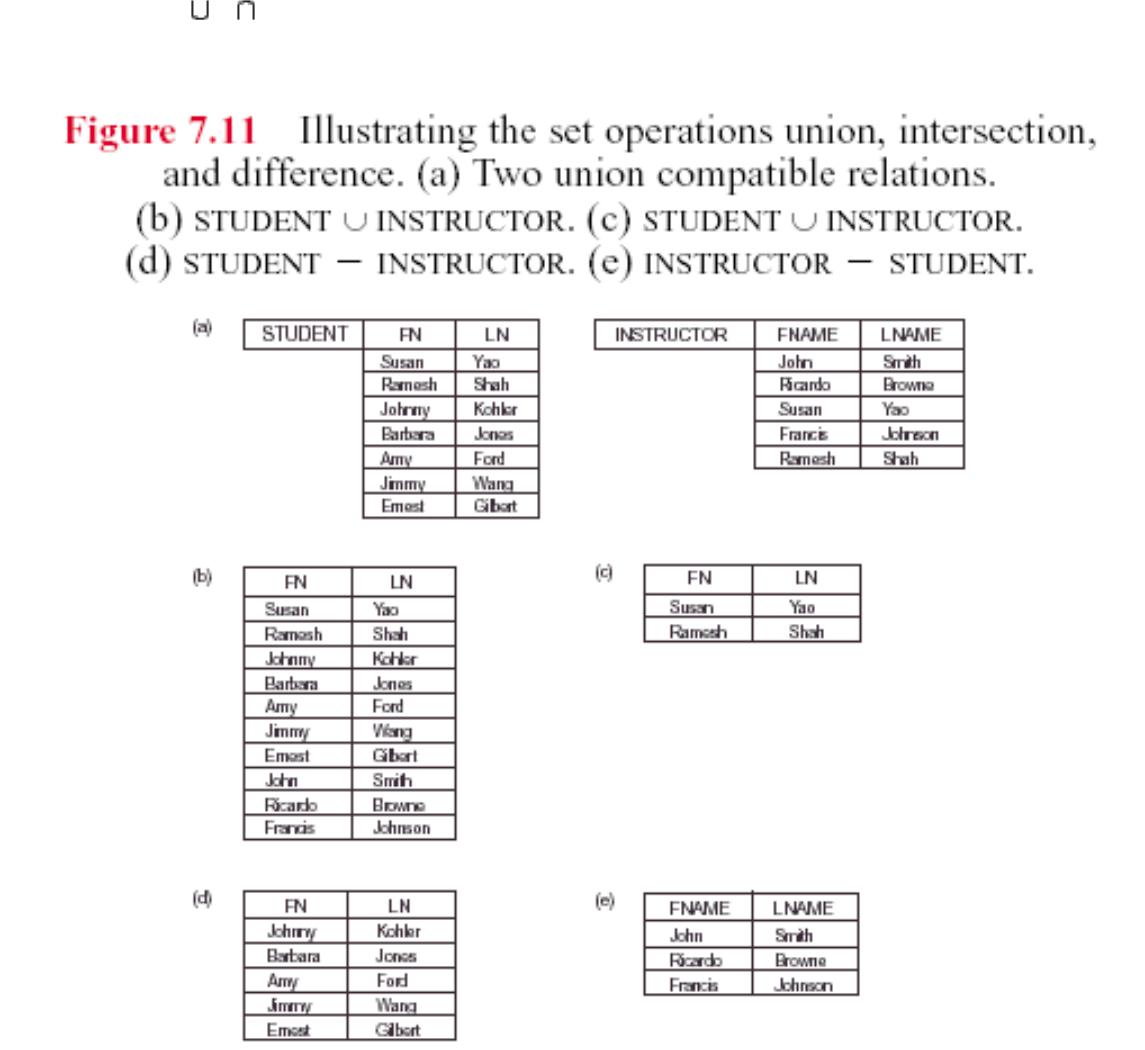
**UNION**: R1 R2,

**INTERSECTION**: R1 R2,

**SET DIFFERENCE**: R1- R2,

**CARTESIAN PRODUCT**: R1X R2.

For , , -, 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. The resulting relation for , , or - has the same attribute names as the first operand relation R1 (by convention).



**CARTESIAN PRODUCT**

R(A1, A2, ..., Am, B1, B2, ..., Bn) R1(A1, A2, ..., Am) X R2 (B1, B2, ..., Bn)

A tuple t exists in R for each combination of tuples t1 from R1 and

t2 from R2 such that:

t[A1, A2, ..., Am] = t1 and t[B1, B2, ..., Bn] = t2

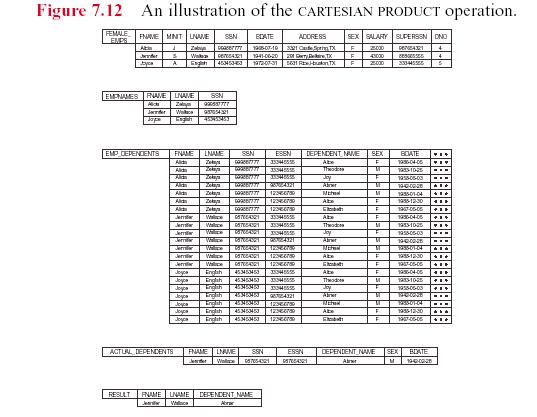
If R1 has n1 tuples and R2 has n2 tuples, then R will have n1\*n2 tuples.

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  MGRSSN=SSN(DEP\_EMP)



**2.6 JOIN Operations**

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

R(A1, A2, ..., Am, B1, B2, ..., Bn) R1(A1, A2, ..., Am) c R2 (B1, B2, ..., Bn)

**EQUIJOIN**:The join condition c includes one or more*equality comparisons* involving attributes from R1 and R2. That is, c is of the form:

(Ai=Bj) AND ... AND (Ah=Bk); 1<i,h<m, 1<j,k<n

In the above EQUIJOIN operation:

Ai, ..., Ah are called the **join attributes** of R1

Bj, ..., Bk are called the **join attributes** of R2

Example of using EQUIJOIN:

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

T DEPARTMENT MGRSSN = SSN EMPLOYEE

RESULT  DNAME,FNAME,LNAME (T)



**NATURAL JOIN** (\*):

In an EQUIJOIN R R1 c R2, the join attribute of R2 appear *redundantly* in the result relation R. In a NATURAL JOIN, the *redundant join attributes* of R2 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 EMPLOYEE \*(DNO),(DNUMBER) DEPARTMENT

RESULT 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:

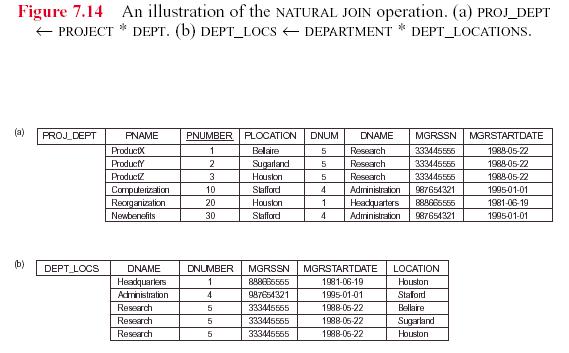
SUPERVISOR(SUPERSSN,SFN,SLN) 

 (EMPLOYEE)

SSN,FNAME,LNAM

TEMPLOYEE \* SUPERVISOR

RESULT 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 |
| two relations. For example: | | | |  |
|  | JOIN ATTRIBUTES | |  |  |
|  | RELATIONSHIP | | |  |
|  |  |  | |  |
| EMPLOYEE.SSN= | | | | EMPLOYEE *manages* |
|  |  |  |  | DEPARTMENT.MGRSSN |
|  | the DEPARTMENT | | |  |
| EMPLOYEE.DNO= | | | | EMPLOYEE *works for* |
|  | DEPARTMENT.DNUMBER | | | the DEPARTMENT |

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

T EMPLOYEE DNO=DNUMBER DEPARTMENT

RESULT  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) SSN,FNAME,LNAME(EMPLOYEE)

TEMPLOYEE SUPERSSN=SSSNSUPERVISOR

RESULT  FNAME,LNAME,SFN,SLN(T)



**Complete Set of Relational Algebra Operations**:

All the operations discussed so far can be described as a sequence of *only* the operations SELECT, PROJECT, UNION, SET DIFFERENCE, and CARTESIAN PRODUCT.

Hence, the set { , , , - , X } is called a *complete set* of relational algebra operations. Any query language *equivalent to* these operations is called **relationally complete**.

For database applications, additional operations are needed that were not part of the *original* relational algebra. These include:

1. Aggregate functions and grouping.
2. OUTER JOIN and OUTER UNION.

**AGGREGATE FUNCTIONS ()**

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

<grouping attributes> <function list>(R)

The grouping attributes are optional

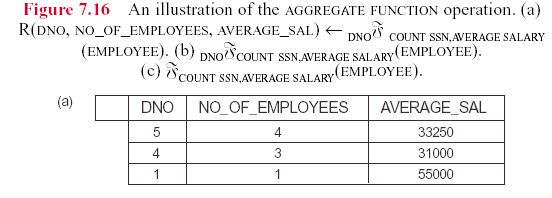
Example 1: Retrieve the average salary of all employees (no grouping needed):

(AVGSAL)AVERAGE SALARY (EMPLOYEE)

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

(DNO,NUMEMPS,AVGSAL)  DNO COUNT SSN, AVERAGE SALARY (EMPLOYEE)

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

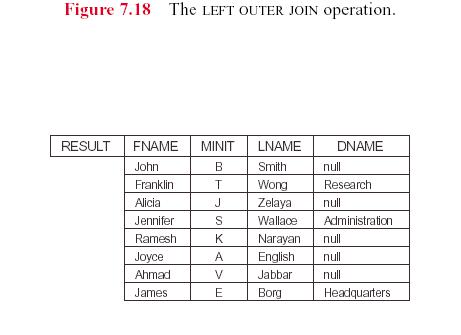


**OUTER JOIN**

In a regular EQUIJOIN or NATURAL JOIN operation, tuples in R1 or R2 that do not have matching tuples in the other relation *do not appear in the result*

|  |  |  |  |
| --- | --- | --- | --- |
|  | Some queries require all tuples in R1 (or R2 | | or both) to appear in |
| the result |  |  |  |
|  | When | no matching tuples are found, **null**s | are placed for the |
| missing attributes |  |  |  |
| **LEFT OUTER JOIN**: R1 | X R2 | lets every tuple in R1 appear in the result | |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |
| --- | --- | --- |
| **RIGHT OUTER JOIN**: R1X | R2 | lets every tuple in R2 appear in the result |
| **FULL OUTER JOIN**: R1X | R2 | lets every tuple in R1 or R2 appear in the result |



**2.8 Examples of Queries in Relational Algebra**

* **Q1: Retrieve the name and address of all employees who work for the ‘Research’ department.**

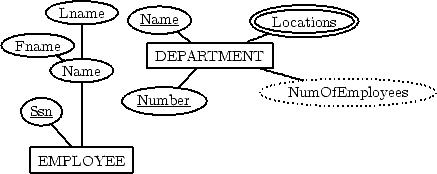
RESEARCH\_DEPT ****DNAME=‘Research‘ (DEPARTMENT)

|  |  |
| --- | --- |
| RESEARCH\_EMPS(RESEARCH\_DEPT | DNUMBER= |
| DNOEMPLOYEEEMPLOYEE) |  |
| RESULT FNAME, LNAME, ADDRESS (RESEARCH\_EMPS) |  |
|  **Q6: Retrieve the names of employees who have no dependents.** |  |
| ALL\_EMPS SSN(EMPLOYEE) |  |
| EMPS\_WITH\_DEPS(SSN) ESSN(DEPENDENT) |  |
| EMPS\_WITHOUT\_DEPS (ALL\_EMPS - EMPS\_WITH\_DEPS) |  |
|  |  |
|  |  |
|  |  |

RESULT LNAME, FNAME (EMPS\_WITHOUT\_DEPS \* EMPLOYEE)

**3.9 Relational Database Design Using ER-to-Relational Mapping**

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.



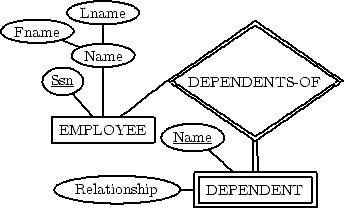
EMPLOYEE

SSN Lname Fname

DEPARTMENT

NUMBERNAME

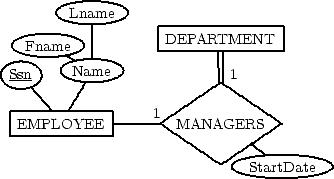
Step 2: For each **weak entity type** W in the ER schema with owner entity type E, create a relation R, and include all simple attributes (or simple components of composite attributes) of W as attributes. In addition, include as foreign key attributes of R the primary key attribute(s) of the relation(s)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



DEPENDENT

EMPL-SSN 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, say S, and include the primary key of T as a foreign key in S. Include all the simple attributes of R as attributes of S.

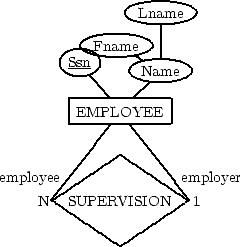


DEPARTMENT

MANAGER-SSN StartDate

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

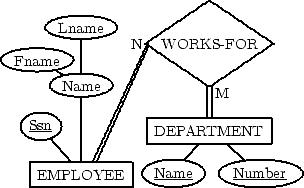
Include the primary key of T as a foreign key of S. Simple attributes of R map to attributes of S.



EMPLOYEE

SupervisorSSN

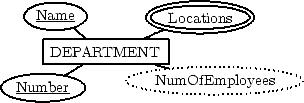
Step 5: For each **binary M:N relationship type** R, create a relation S. Include the primary keys of participant relations as foreign keys in S. Their combination will be the primary key for S. Simple attributes of R become attributes of S.



WORKS-FOR

EmployeeSSN DeptNumber

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 A and K.



DEP-LOCATION

Location DEP-NUMBER

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 > 1.

For a recursive relationship, we will need a new relation.

**Questions**

1. Define the following terms with an example for each.
2. Explain:

3. i) Domain constraint ii) Semantic integrity constraint iii) Functional

dependency constraint

1. List the characteristics of relation? Discuss any one?
2. Discuss various types of Inner Join Operations?
3. Discuss the characteristics of a relation, with an example
4. Briefly discuss the different types of update operations on relational database. show an example of
5. What is valid state and an invalid state,with respect to a database
6. Define referential integrity constraint. Explain the importance of referential integrity constraint. How is this constraint implemented in SQL
7. Define referential integrity in each of the update operation