**UNIT-3**

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

**OUTLINE**

Relational Model Concepts, Domains, Attributes, and Relations, Characteristics of Relations, Relational Model Constrains and Relational Database Schemas, Codd’s Rule

**Introduction**

The principles of the relational model were first outlined by Dr. E. F. Codd in a June 1970 paper called "A Relational Model of Data for Large Shared Data Banks:' In this paper. Dr. Codd proposed the relational model for database systems. The more popular models used at that time were hierarchical and network, or even simple flat file data structures. Relational database management systems (RDBMS) soon became very popular, especially for their ease of use and flexibility in structure. In addition, a number of innovative vendors, such as Oracle, supplemented the RDBMS \with a suite of powerful application development and user products, providing a total solution.

Earlier we saw how to convert an unorganized text description of information requirements into a conceptual design, by the use of ER diagrams. The advantage of ER diagrams is that they force us to identify data requirements that are implicitly known, but not explicitly written down in the original description. Here we will see how to convert this ER into a **logical design** of a relational database. The logical model is also called a **Relational Model**.

**Relational Model Concepts**

* The relational Model of Data is based on the concept of a Relation.
* A Relation is a mathematical concept based on the ideas of sets.
* The strength of the relational approach to data management comes from the formal foundation provided by the theory of relations.
* The model was first proposed by Dr. E.F. Codd of IBM Research in 1970.
* The relational model represents the database as a collection of **relations.**

**RELATION**

**Informal Definition**

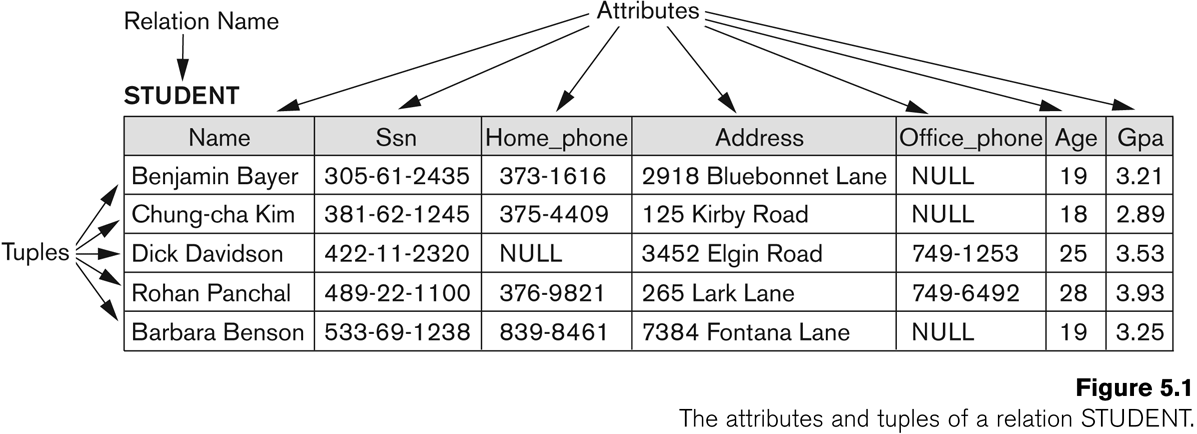
* Each **relation** resembles a table of values.
  + A relation may be thought of as a **set of rows**.
  + A relation may alternately be thought of as a **set of columns**.
  + Each row represents a fact that corresponds to a real-world **entity** or **relationship**.
  + Each row has a value of an item or set of items that uniquely identifies that row in the table.
  + Sometimes row-ids or sequential numbers are assigned to identify the rows in the table.
  + Each column typically is called by its column name or column header or attribute name.

**Formal Definitions**

In the formal relational model terminology

* The table is called **a *relation.***
* A row is called a ***tuple.***
* A column header is called an ***attribute.***
* The **data type** describing the types of values that can appear in each column is represented by a ***domain***of possible values.

**Example of STUDENT Relation (see figure below)**



**Relational Schema:** A relation schema R, denoted by **R( A1, A2, A3, …, An** ) is made up of a relation name R and a list of attributes **A1, A2, A3, …, An** .

**Attribute**: It is the name of a column in a particular table (all data is stored in tables). Each attribute Ai must have a *domain*, dom(Ai).

**Domain**: It is a set of atomic values that an attribute can take and is denoted by dom(A). Atomic means that each value in the domain is indivisible as far as the relational model is concerned.

**Degree/Arity of a Relation**: The degree (or **arity**) of a relation is the number of attributes ***n*** of its relation schema.

An **example** of a relation schema for a relation of **degree seven**, which describes university students, is the following:

**STUDENT** (Name, SSN, HomePhone, Address, OfficePhone, Age, GPA)

Some domains for some of the attributes of the STUDENT relation:

dom(Name)={all Possible values correspondence to attribute name separated by ,(comma)};

dom(SSN)={ all Possible values correspondence to attribute Social\_security\_numbers separated by ,(comma)};

It is also possible to refer to attributes of a relation schema by their position within the relation; thus, the second attribute of the STUDENT relation is SSN, whereas the fourth attribute is Address.

Formally, Given R(A1, A2, .........., An)

**r(R) ⊂ dom (A1) X dom (A2) X ....X dom(An)**

**Relation state:**

A relation (or relation state) r of the relation schema R(A1, A2, .........., An) also denoted by *r(R),* **is a *set of tuples* (rows)**

* + r(R) = {t1, t2, …, tn} where each ti is an n-tuple.
  + ti = <v1, v2, …, vn> where each vj *element-of* dom(Aj).

Let R (A1, A2) be a relation schema:

Let dom(A1) = {0,1} and dom(A2) = {a,b,c}, Then,

dom(A1) X dom(A2) is all possible combinations:

**{<0,a> , <0,b> , <0,c>, <1,a>, <1,b>, <1,c> }**

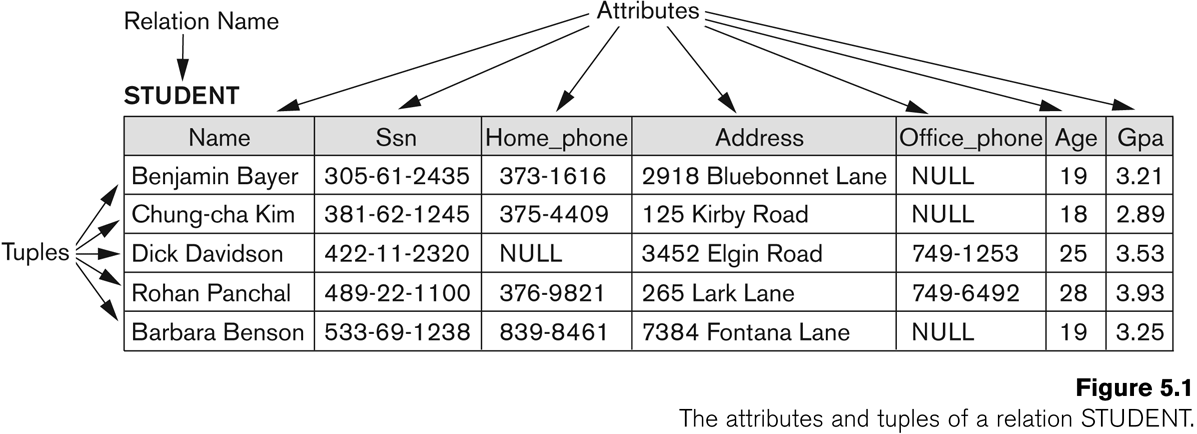
**The relation state r(R) ⊂ dom(A1) X dom(A2)**

**Definition Summary**

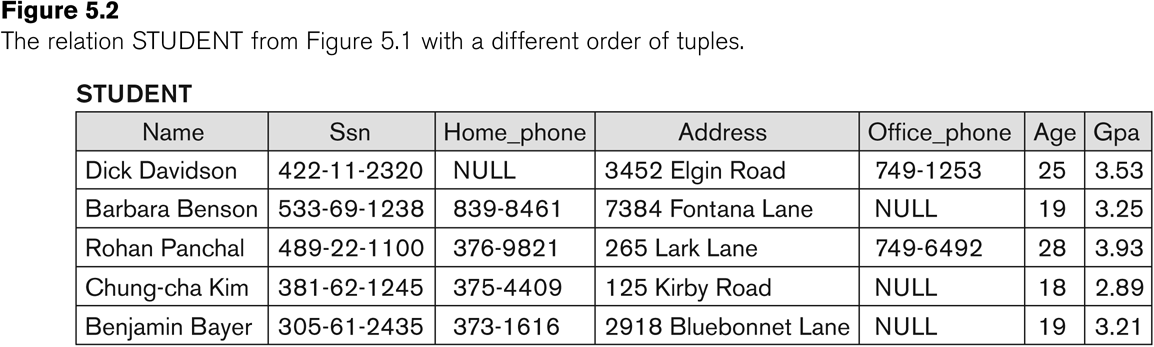
|  |  |
| --- | --- |
| **Informal Terms** | **Formal Terms** |
| **Table** | Relation |
| **Column Header** | Attribute |
| **All possible Column Values** | Domain |
| **Row** | Tuple |
| **Table Definition** | Schema of a Relation |
| **Populated Table** | State of the Relation |

**Characteristics Of Relations**

1. **Ordering of tuples in a relation r(R)**: The tuples are *not* considered to be ordered, even though they appear to be in the tabular form.
2. **Ordering of attributes in a relation schema R** (and of values within each tuple): We will consider the attributes in R(A1, A2, ..., An) and the values in t=<v1, v2, ..., vn> to be *ordered* .
3. **Values in a tuple**: All values are considered *atomic* (indivisible). A special **null** value is used to represent values that are unknown or inapplicable to certain tuples.



The figure below shows the same state as previous Figure (but with different order of tuples)



4. Values and NULLs in tuples

* + Each value in a tuple is atomic ie a single value. That’s why it is called **Flat relational model**
    - Composite and multi valued attributes are not allowed.
  + Multi valued attributes
    - Must be represented by separate relations
  + Composite attributes
    - Represented only by simple component attributes in basic relational model
* NULL values
  + Represent the values of attributes that may be unknown or may not apply to a tuple
  + Meanings for NULL values
    - *Value unknown*
    - *Value exists but is not available*
    - *Attribute does not apply to this tuple (also known as value undefined)*

**Relational keys**

There are two kinds of keys in relations. The first are identifying keys: the **primary key** is the main concept, while two other keys – **super key** and **candidate key** – are related concepts.

The second kind is the foreign key.

**Identifying Keys:**

**Super Keys** A super key is a set of one or more attributes whose values can be used to uniquely identify a tuple within a relation. A relation may have more than one super key, but it always has at least one: the set of all attributes that make up the relation.

**Candidate Keys** A candidate key is a super key that is minimal; that is, there is no proper subset that is itself a superkey. A relation may have more than one candidate key, and the different candidate keys may have a different number of attributes.

A candidate key has two properties:

(i) in each tuple of R, the values of K uniquely identify that tuple (uniqueness)

(ii) no proper subset of K has the uniqueness property (irreducibility).

**Primary Key** The primary key of a relation is a candidate key especially selected to be the key for the relation. In other words, it is a choice, and there can be only one candidate key designated to be the primary key.

**Relationship between identity keys**

The relationship between keys:

**Superkey ⊇ Candidate Key ⊇ Primary Key**

**Foreign keys** The attribute(s) within one relation that matches a candidate key of another relation. A relation may have several foreign keys, associated with different target relations.

**Relational Model Notations:**

* A relation schema R of degree n is denoted by R(A1, A2, . . ., An).
* The letters Q, R, S denote relation names.
* The letters q, r, s denote relation states.
* The letters t, u, v denote tuples.
* In general, the name of a relation schema such as STUDENT also indicates the current set of tuples in that relation—the current relation state—whereas STUDENT(Name, SSN, . . .) refers only to the relation schema.
* An attribute A can be qualified with the relation name R to which it belongs by using the dot notation R.A—for example, STUDENT.Name or STUDENT.Age. This is because the same name may be used for two attributes in different relations. However, all attribute names in a particular relation must be distinct.
* An n-tuple t in a relation r(R) is denoted by t = <v1, v2, . . ., vn>, where vi is the value corresponding to attribute Ai. The following notation refers to component values of tuples:
* Both t[Ai] and t.Ai refer to the value vi in t for attribute Ai.
* Both t[Au, Aw, . . ., Az] and t.(Au, Aw, . . ., Az), where Au, Aw, . . ., Az is a list of attributes from R, refer to the subtuple of values <vu, vw, . . ., vz> from t corresponding to the attributes specified in the list.

**RELATIONAL MODEL CONSTRAINTS AND RELATIONAL DATABASE SCHEMAS**

**Constraints:** Constraints are ***conditions*** / ***rules*** that must hold on *all* valid relation instances.

* Restrictions on the actual values in a database state.
* Derived from the rules in the miniworld that the database represents.

**Domain constraints**

Domain constraints specify that within each tuple, the value of each attribute A must be an atomic value from the domain dom(A). The data types associated with domains typically include:

* + Numeric data types for integers and real numbers
  + Characters
  + Booleans
  + Fixed-length strings
  + Variable-length strings
  + Date, time, timestamp
  + Other special data types

**Key Constraints and Constraints on Null Values**

**Superkey of R:** A set of attributes, SK, of R such that no two tuples in any valid relational instance, r( R), will have the same value for SK. Therefore, for any two distinct tuples, t1 and t2 in r( R),

t1[ SK] != t2[SK].

**Key of R:** A minimal superkey. That is, a superkey, K, of R such that the removal of ANY attribute from K will result in a set of attributes that are not a superkey.

Example: The STUDENT relation schema:

**STUDENT**(Rollno, Name, Age, Address, Grade, Phoneno) has several candidate keys

**Key1** = {Rollno}, **Key2** = {Name, Address}, **Key3** = {Name, Phoneno} etc., which are also superkeys.

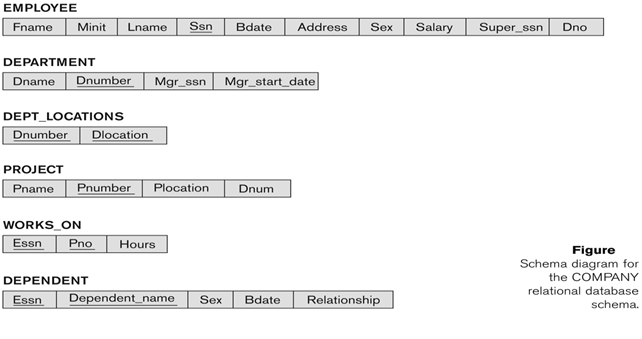
If a relation has *several* candidate keys, one is chosen arbitrarily to be the primary key. The primary key attributes are *underlined*.

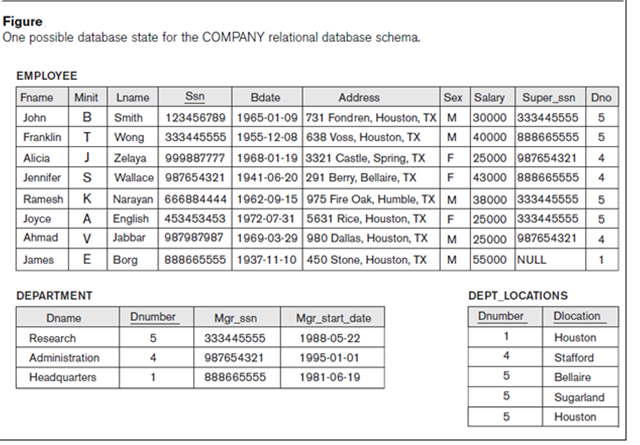
**Relational Databases and Relational Database Schema**

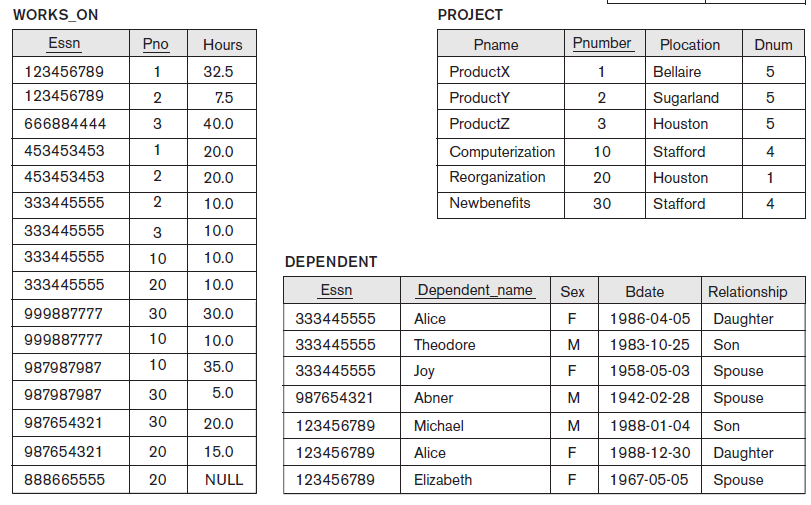
* **Relational database schema** *S*
  + Set of relation schemas *S =* {*R*1, *R*2, ..., *Rm*}
  + Set of integrity constraints IC
* **Relational database state**
  + Set of relation states *DB =* {*r1*, *r2*, ..., *rm*}
  + Each *ri* is a state of *Ri* and such that the *ri*relation states satisfy integrity constraints specified in IC
* **Invalid state**
  + Does not obey all the integrity constraints
* **Valid state**
  + Satisfies all the constraints in the defined set of integrity constraints IC

In figures below the relational database schema of COMPANY as well as the Relational database state of COMPANY are shown.

COMPANY={EMPLOYEE, DEPARTMENT, DEPT\_LOCATIONS, PROJECT, WORKS\_ON, DEPENDENT}.



****

****

**Integrity constraints**

Most database applications have certain integrity constraints that must hold for the data. The simplest type of integrity constraint involves specifying a data type for each data item. Integrity constraints can be classified as:

1. **Entity Integrity Constraint**
2. **Referential Integrity Constraint**

**I. Entity Integrity Constraint**

* This type of constraint is specified on individual relation.
* The Entity integrity constraints states that no primary key value can be NULL because the primary key value is used to distinguish each tuples in a relation.

**II. Referential Integrity Constraints**

* This constraint is specified between *two* relations.
* This constraint establishes a relationship across two relations: the **master or** **referencing relation** and a **detail or** **referenced relation**.
* The relationship ensures that
  + - The records cannot be inserted into a table if corresponding records in master table do not exist.

The records of the master table cannot be deleted if corresponding records in detail table exist.

* Tuples in the ***referencing relation*** R1 have attributes FK (called foreign key attributes) that reference the primary key attributes PK of the ***referenced relation*** R2. A tuple t1 in R1 is said to reference a tuple t2 in R2 if t1[FK] = t2[PK].
* A referential integrity constraint can be displayed in a relational database schema as a directed arc from R1.FK to R2.

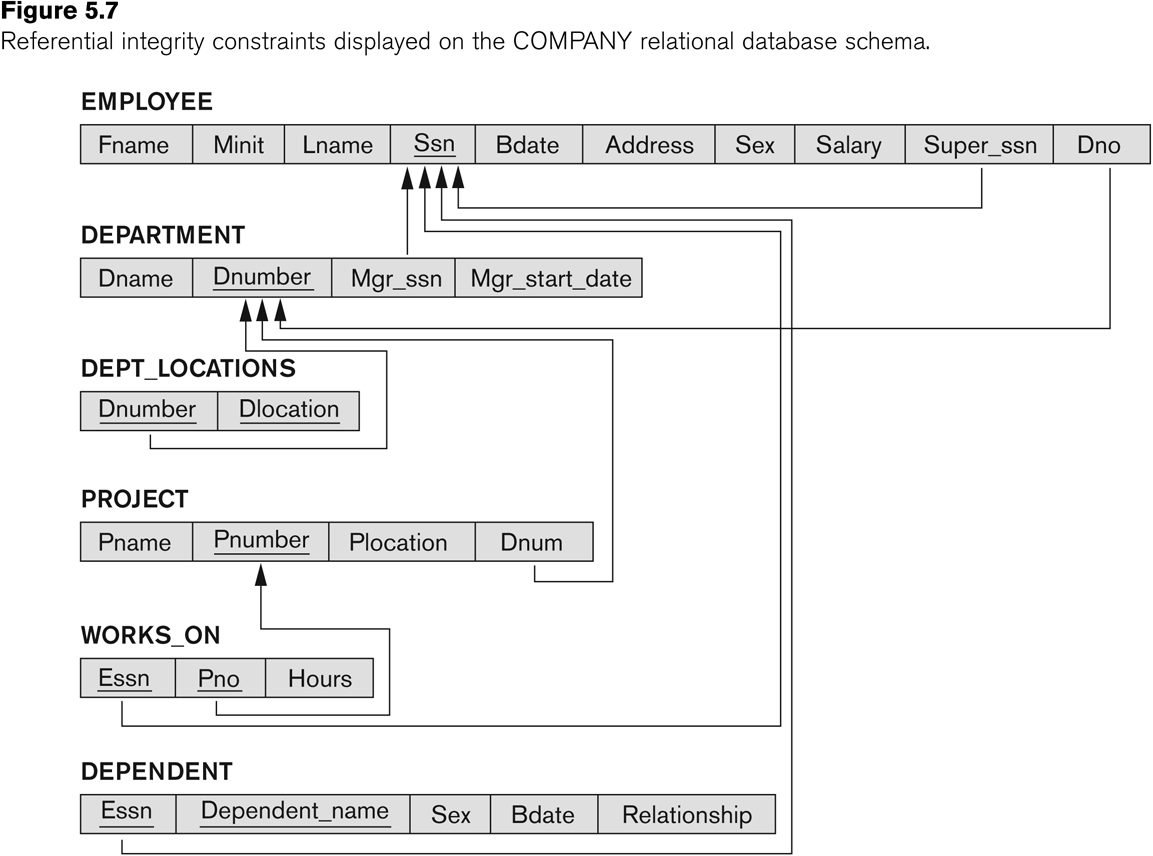
**Foreign key** rules:

* The attributes in FK have the same domain(s) as the primary key attributes PK.
* The value in the foreign key column (or columns) FK of the the **referencing relation** R1 can be either:

i) A value of an existing primary key value of the corresponding primary key PK in the **referenced relation** R2, or

ii) A **null**.

In case (ii), the FK in R1 should not be a part of its own primary key.



**Codd's 12 rules**

**Dr. E. F. Codd's 12 rules for defining a fully relational database**

A **relational database management system (RDBMS)** is a database management system (DBMS) that is based on the relational model as introduced by *E. F. Codd*. Most popular commercial and open source databases currently in use are based on the relational model.

A short definition of an RDBMS may be a DBMS in which data is stored***in the form of tables and the relationship among the data is also stored in the form of tables.***

***E.F. Codd*,** the famous mathematician has introduced 12 rules for the relational model for databases commonly known as **Codd's rules**. The rules mainly define what is required for a DBMS for it to be considered *relational*, i.e., an RDBMS. The rules and their description are as follows:-

**Rule 0: Foundation Rule**

A *relational database management system*should be capable of using its *relational* facilities (exclusively) to *manage* the *database*.

**Rule 1: Information Rule**

All information in the database is to be represented in one and only one way. This is achieved by values in column positions within rows of tables.

**Rule 2: Guaranteed Access Rule**

All data must be accessible with no ambiguity, that is, Each and every datum (atomic value) is guaranteed to be logically accessible by resorting to a combination of table name, primary key value and column name.

**Rule 3: Systematic treatment of null values**

Null values (distinct from empty character string or a string of blank characters and distinct from zero or any other number) are supported in the fully relational DBMS for representing missing information in a systematic way, independent of data type.

**Rule 4: Dynamic On-line Catalog Based on the Relational Model**

The database description is represented at the logical level in the same way as ordinary data, so authorized users can apply the same relational language to its interrogation as they apply to regular data. The authorized users can access the database structure by using common language i.e. SQL.

**Rule 5: Comprehensive Data Sublanguage Rule**

A relational system may support several languages and various modes of terminal use. However, there must be at least one language whose statements are expressible, per some well-defined syntax, as character strings and whose ability to support all of the following is comprehensible:

1. data definition
2. view definition
3. data manipulation (interactive and by program)
4. integrity constraints
5. authorization
6. Transaction boundaries (begin, commit, and rollback).

**Rule 6: View Updating Rule**

All views that are theoretically updateable are also updateable by the system.

**Rule 7:  High-level Insert, Update, and Delete**

The system is able to insert, update and delete operations fully. It can also perform the operations on multiple rows simultaneously.

**Rule 8: Physical Data Independence**

Application programs and terminal activities remain logically unimpaired whenever any changes are made in either storage representation or access methods.

**Rule 9: Logical Data Independence**  
Application programs and terminal activities remain logically unimpaired when information preserving changes of any kind that theoretically permit unimpairment are made to the base tables.

**Rule 10: Integrity Independence**  
Integrity constraints specific to a particular relational database must be definable in the relational data sublanguage and storable in the catalog, not in the application programs.

**Rule 11: Distribution Independence**  
The data manipulation sublanguage of a relational DBMS must enable application programs and terminal activities to remain logically unimpaired whether and whenever data are physically centralized or distributed.

**Rule 12: Nonsubversion Rule**  
If a relational system has or supports a low-level (single-record-at-a-time) language, that low-level language cannot be used to subvert or bypass the integrity rules or constraints expressed in the higher-level (multiple-records-at-a-time) relational language.

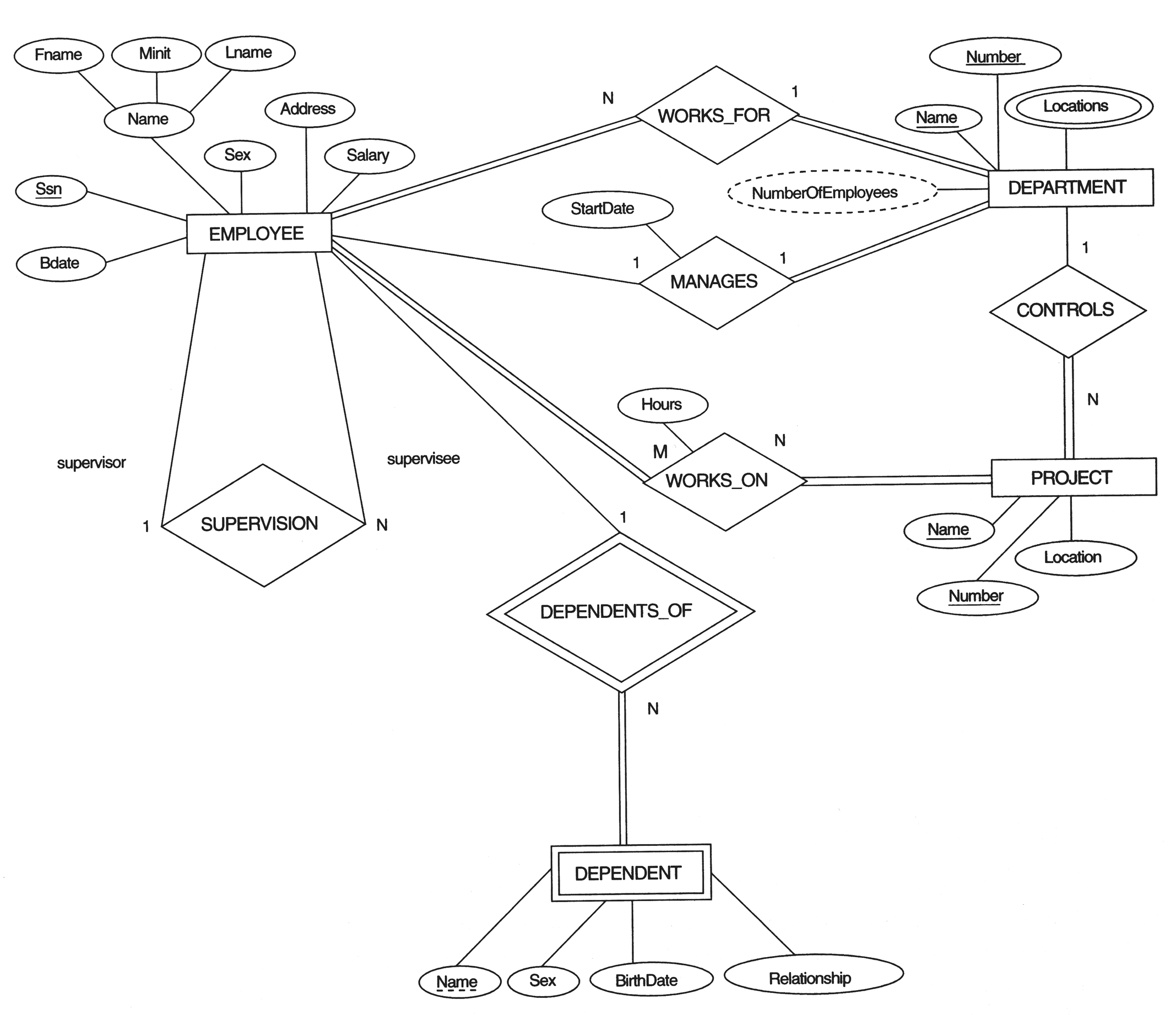
**ER to relational Mapping:** Data base design using ER to relational language.

In this chapter we learn how to map a conceptual schema deisgn in the ER model to a relational database schema. An algorithm for ER-to-relational mapping was given and illustrated by examples from the COMPANY database. In a relational schema, relationship types are not represented explicitly.

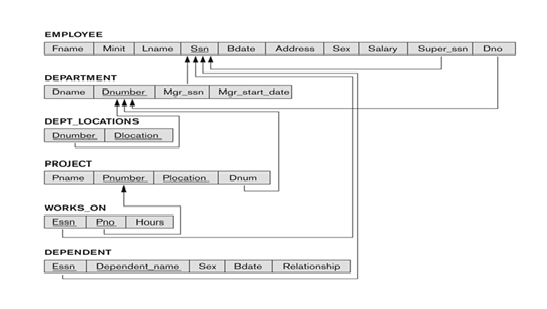
**ER-to-Relational Mapping Algorithm**

* Step 1: Mapping of Regular Entity Types
* Step 2: Mapping of Weak Entity Types
* Step 3: Mapping of Binary 1:1 Relation Types
* Step 4: Mapping of Binary 1:N Relationship Types.
* Step 5: Mapping of Binary M:N Relationship Types.
* Step 6: Mapping of Multivalued attributes.
* Step 7: Mapping of N-ary Relationship Types.

Consider the ER MODEL of COMPANY Database:



Result of mapping the COMPANY ER schema into a relational schema after applying the above algorithm:

****

**Step 1: Mapping of Regular/Strong Entity Types.**

* Need to create a new relation.
* For each regular (strong) entity type E in the ER schema, create a relation R that includes all the simple attributes of E. If there is composite attribute, then select the simple attributes associated with that composite attribute.
* Choose one of the key attributes of E as the primary key for R.

Hence, after step 1, the strong entities converted to the following relations:

EMPLOYEE (ssn, fname, minit, lname, bdate, age, sex, address)

DEPARTMENT (dnumber, dname) As location is multi-valued, we will consider in step 6.

PROJECT(pnumber, pname, location)

**Step 2: Mapping of Weak Entity Types**

* Need to create a new relation.
* For each weak entity type W in the ER schema with owner entity type E, create a relation R & include all simple attributes (or simple components of composite attributes) of W as attributes of R.
* Also, include as foreign key attributes of R the primary key attribute(s) of the relation(s) that correspond to the owner entity type(s).
* 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.

**Example:** Create the relation DEPENDENT in this step to correspond to the weak entity type DEPENDENT.

* Include the primary key SSN of the EMPLOYEE relation as a foreign key attribute of DEPENDENT (renamed to ESSN).
* The primary key of the DEPENDENT relation is the combination {ESSN, DEPENDENT\_NAME} because DEPENDENT\_NAME is the partial key of DEPENDENT.

DEPENDENT(essn, dependent\_name , sex, bdate, relationship)

**Step 3: Mapping of Binary 1:1 Relationship Types**

* Identify the 1:1 relationship.
* NO need to create a new relation.
* For each 1:1 relationship type R, **identify the relations** S and T that participating in R.
* **Apply Foreign key approach** Choose one relation S. Include the primary key of T as a foreign key in S.
* It is better to choose an entity type with total participation in R in the role of S.

**Example:** We found MANAGES is the 1:1 relationship type. No relation is created for the 1:1 relationship type MANAGES. Select DEPARTMENT as S. Include the primary key Ssn of the EMPLOYEE relation as a foreign key attribute of DEPARTMENT (renamed to Mgr\_ssn). Include the attribute Start\_date of the MANAGES relationship in DEPARTMENT and rename it Mgr\_start\_date. Hence after this step, the changes in the relation DEPARTMENT is as:

DEPARTMENT (dnumber, dname, mgr\_ssn, start\_date)

**Step 4: Mapping of Binary 1:N Relationship Types**

* Identify the 1:N relationship.
* Need to create a new relation.
* For each binary 1:N relationship type R, **identify the relation S** that represents the participating entity type at the N-side of the relationship. Let T denotes the other participating entity.
* Include the primary key of T as a foreign key in S.
* Include any simple attributes of the relationship as attributes of S.

**Example:** We identified **WORKS\_FOR, CONTROLS & SUPERVISION** are the 1:N relationship types

**WORKS\_FOR: N-Side is EMPLOYEE as S & DEPARTMENT as T**

* Include the primary key Dnumber of DEPARTMENT as foreign key in EMPLOYEE. Call it Dno.

**SUPERVISION: N-Side is EMPLOYEE only**

* Include the primary key of EMPOYEE as foreign key in EMPLOYEE. Call it Super\_ssn.

**CONTROLS: N-Side is PROJECT & DEPARTMENT as T**

* Include the primary key Dnumber of DEPARTMENT in PROJECT. Call it Dnum.

**Hence after step 4, the changes in the relations are:**

EMPLOYEE (ssn, fname, minit, lname, bdate, age, sex, address, dnumber, super\_ssn)

PROJECT(pnumber, pname, location, dnumber)

**Step 5: Mapping of Binary M:N Relationship Types**

* Need to create a new relation.
* For each binary M:N relationship type R, **create a new relation S**.
* Include the primary keys of both participating entities as foreign keys.
* Their combination will form the primary key of S.
* Also include any simple attributes of R.

**Example:** Create the relation WORKS\_ON. Include the primary keys of PROJECT and EMPLOYEE as foreign keys. Rename them Pno and Essn. Also include an attribute Hours. The primary key is the combination of {Essn, Pno}.

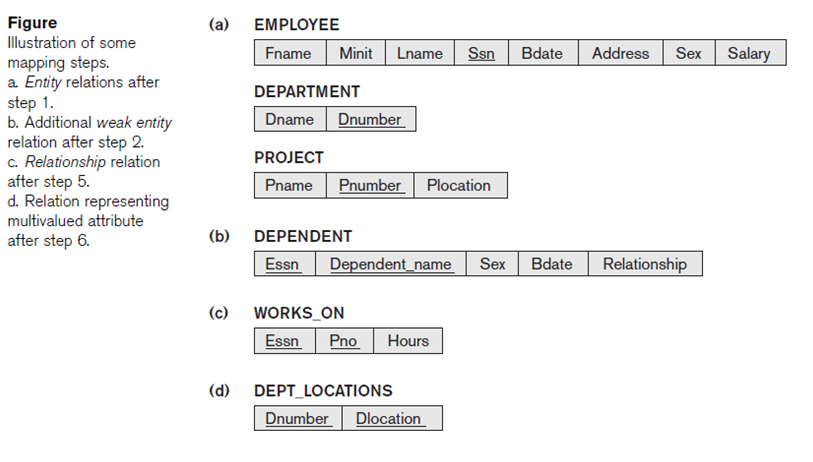
**WORKS\_ON(essn, pno, hours)**

**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 of relationship type that has A as an attribute.
* The primary key of R is the combination of A and K. If the multivalued attribute is composite, we include its simple components.

**Example:** The relation DEPT\_LOCATIONS is created. The attribute DLOCATION represents the multivalued attribute Locations of DEPARTMENT, while Dnumber (as foreign key) representing the primary key of the DEPARTMENT relation. The primary key of R is the combination of {Dnumber, Dlocation}.

DEPT\_LOCATION(dnumber, dlocation)

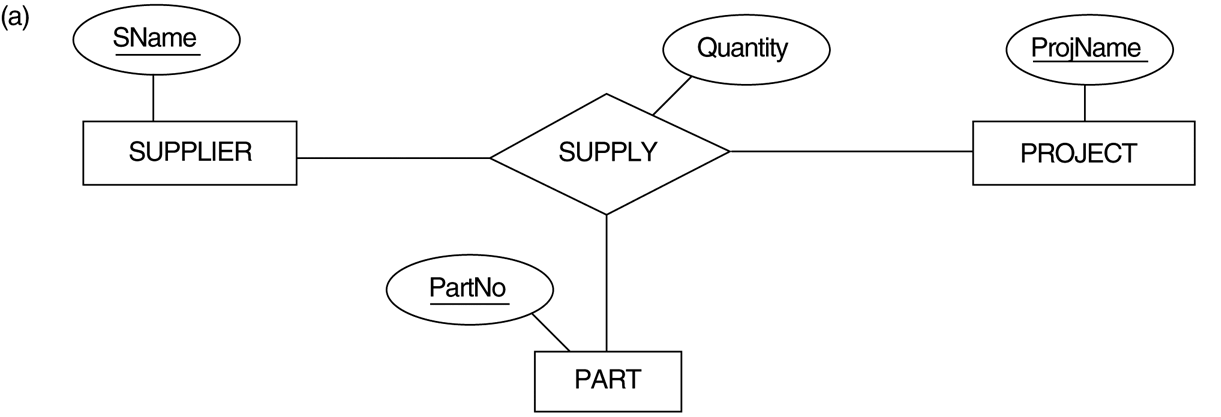
****

**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.

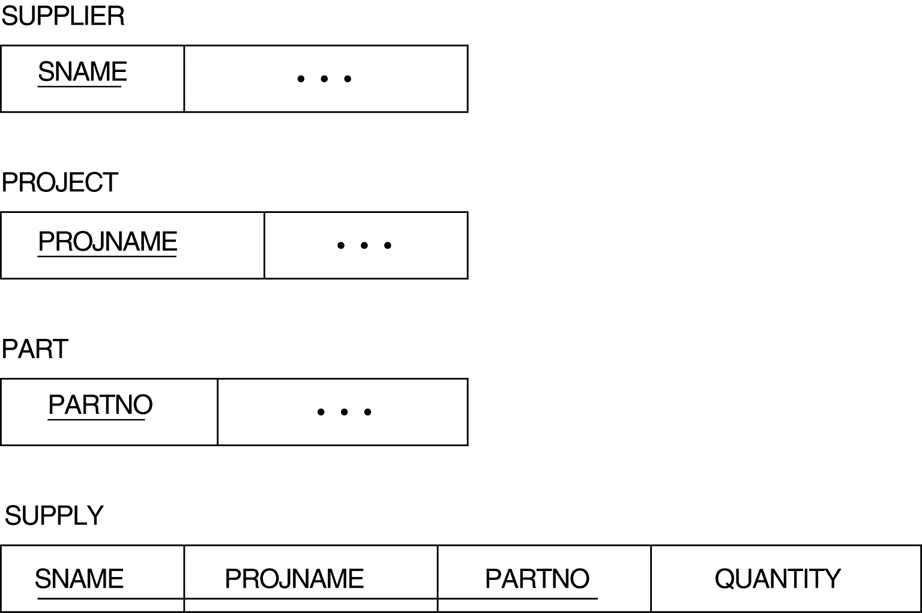
**Example:** The relationship type SUPPY in the ER-diagram on the next slide. This can be mapped to the relation SUPPLY shown in the relational schema, whose primary key is the combination of the three foreign keys {Sname, Part\_no, Proj\_name}

**Ternary relationship types (a) The SUPPLY relationship.**



SUPPLY(sname, projname, partno, quantity)

Mapping the *n*-ary relationship type SUPPLY from above figure



**Relational Algebra**

**OUTLINE:** Basic operations selection and projection, Set Theoretic operations Union, Intersection, set difference and division, Join operations: Inner, Outer: Left outer, Right outer and full outer join.

**Relational Algebra**

Relational Algebra is a **procedural query language** which consists of a set of operations that take one or two relations as input and produce a new relation as their result.

Relational Algebra consists of several groups of operations

1. **Unary Relational Operations**

* SELECT (symbol: **σ**  (sigma))
* PROJECT (symbol: **Π** (pi))

1. **Relational Algebra Operations From Set Theory**

* UNION ( **U** ), INTERSECTION (**∩**), DIFFERENCE (or MINUS, **–** )
* CARTESIAN PRODUCT ( **x** )

1. **Binary Relational Operations**

* JOIN (several variations of JOIN exist)
* DIVISION

1. **Additional Relational Operations**

* OUTER JOINS

1. **Unary Relational Operation**
2. **SELECT Operation**

* It is a unary operation defined on a single relation.
* It is denoted as **σ**.
* In general, the select operation is denoted by **σ P(R)** where R is the input relation, P is the predicate or condition to be checked is a Boolean expression and the symbol **σ** (sigma) is used to denote the select operator, and the selection condition. We can use the following predicate : **> <, ≤, ≥, ≠, ^ (AND), V (OR).**
* SELECT operation is used to select a *subset* of the tuples from a relation that

satisfy a **selection condition**.

* It is a filter that keeps only those tuples that satisfy a qualifying condition – those

satisfying the condition are selected while others are discarded.

**Example:** To select the EMPLOYEE tuples whose department number is four or those whose salary is greater than 30,000 the following notation is used:

**σ DNO = 4 (EMPLOYEE)**

**σ SALARY > 30,000 (EMPLOYEE)**

**SELECT Operation Properties**

* The SELECT operation **σ P(R)** produces a relation S that has the same schema as R
* The SELECT operation σ is commutative; i.e.,

**σ P1(σ P2 ( R)) = σ P2 (σ P1( R))**

* A cascaded SELECT operation may be applied in any order; i.e.,

**σ P1(σ P2 (σ P3 (R))) = σ P2(σ P3 (σ P1 (R)))**

1. **PROJECT Operation**

* It is a unary operation defined on a single relation
* It is denoted as **Π(PI)**.
* This operation selects certain *columns* from the table and discards the other columns.
* The general form of the project operation is **Π <attribute list>  (R)** where **Π** (pi) is the symbol used to represent the project operation and <attribute list> is the desired list of attributes from the attributes of relation R.
* The project operation *removes any duplicate tuples,* so the result of the project operation is a set of tuples and hence a valid relation.

**Example:** To list each employee’s first and last name and salary, the following is used:

**Π LNAME, FNAME, SALARY(EMPLOYEE)**

**PROJECT Operation Properties**

* The number of tuples in the result of projection **Π** **<list>** (R) is always less or equal to the number of tuples in R.
* If the list of attributes includes a key of R, then the number of tuples is equal to the number of tuples in R.

1. **Relational Algebra Operations From Set Theory**
2. **UNION Operation** 
   * It is a Binary operation, denoted by U
   * The result of R U S, is a relation that includes all tuples that are either in R or in S or in both R and S
   * Duplicate tuples are eliminated
   * **The two operand relations R and S must be “type compatible” (or UNION compatible) if**

* R and S must have same number of attributes
* Each pair of corresponding attributes must be type compatible (have same or

compatible domains)

**Example1: Consider two relational schema R & S**

|  |  |
| --- | --- |
| ID | NAME |
| 101 | JONES |
| 103 | SMITH |
| 104 | LAL |
| 106 | BARON |
| 107 | EVON |
| 110 | DREW |
| 112 | SMITH |

**R S R U S**

|  |  |
| --- | --- |
| ID | NAME |
| 101 | JONES |
| 103 | SMITH |
| 104 | LAL |
| 107 | EVON |
| 110 | DREW |
| 112 | SMITH |

|  |  |
| --- | --- |
| ID | NAME |
| 103 | SMITH |
| 104 | LAL |
| 106 | BARON |
| 110 | DREW |

**Example 2: consider the following relational schema**

Borrower (customer-name, loan-number)  
Depositor (customer-name, account-number)  
Customer (customer-name, street-number, customer-city)

Q. List all the customers who have either an account or a loan or both Code can be represented using Relational Algebra is :

**Π customer-name (Borrower) U Π customer-name (Depositor)**

**Type Compatibility: 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.**

**b. INTERSECTION Operation**

* It is a Binary operation, denoted by **∩**.
* The result of this operation, denoted by R **∩** S, is a relation that includes all tuples that are in both R and S.
* The two operands must be "type compatible"

**Example 1: From the above relational schema R & S, the result of the intersection operation is**

|  |  |
| --- | --- |
| ID | NAME |
| 103 | SMITH |
| 104 | LAL |
| 110 | DREW |

**R ∩ S =**

**Example 2:**

List all the customers who have both a loan and an account from example 2 above.

**Π customer-name (Borrower) ∩ Π customer-name (Depositor)**

**c. Set Difference (or MINUS) Operation**

* SET DIFFERENCE (also called MINUS or EXCEPT) is denoted by –
* The result of R – S, is a relation that includes all tuples that are in R but not in S
* The attribute names in the result will be the same as the attribute names in R
* The two operand relations R and S must be “type compatible”

|  |  |
| --- | --- |
| ID | NAME |
| 101 | JONES |
| 107 | EVON |
| 112 | SMITH |

**Example 1: From the above relational schema R & S, the result of the Set Difference operation is**

**R - S = S - R =**

|  |  |
| --- | --- |
| ID | NAME |
| 106 | BARON |

**Example 2:**Find the names of all customers who have an account but not a loan.

**Π customer-name (Depositor) - Π customer-name (Borrower)**

**Some properties of UNION, INTERSECT, and DIFFERENCE**

* Both union and intersection are ***commutative operations****;* that is

**R U S = S U R, and R ∩ S = S ∩ R**

* Both union and intersection can be treated as n-ary operations applicable to any number of relations as both are ***associative*** *operations;* that is

**R U (S U T) = (R U S) U T, and (R ∩ S) ∩ T = R ∩ (S ∩ T)**

* The minus operation is ***not commutative****;* that is, in general

**R - S ≠ S – R**

1. **CARTESIAN (or cross product) Operation**

* If 2 relations R & S are **product compatible**, then they can participate in a Cartesian product operation.
* **if the** 2 relations don’t have the common attribute names, then the relations are called **product compatible**.
* This operation is used to combine tuples from two relations in a combinatorial fashion.
* Denoted by **R(A1, A2, . . ., An) X S(B1, B2, . . ., Bm)**
* If one relation has K Tuples & M attributes and other has L Tuples & N attributes , the resultant relation Q will contains **(K \* L)** tuples & with degree (**N +M)** attributes, Q(A1, A2, . . ., An, B1, B2, . . ., Bm) in that order.
* The resulting relation state has one tuple for each combination of tuples—one from R and one from S.
* The two operands do NOT have to be "type compatible”

**Example:**

**R S R X S**

|  |  |  |
| --- | --- | --- |
| A | B | C |
| a | a | a |
| b | b | b |
| c | c |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | C | P | Q |
| a | a | a | x | x |
| a | a | a | y | y |
| b | b | b | x | x |
| b | b | b | y | y |
| c | c | c | x | x |
| c | c | c | y | y |

|  |  |
| --- | --- |
| P | Q |
| x | x |
| y | y |

**Total attributes = 5**

**Total no. of tuples = 6**

1. **Binary Relational Operations**

**JOIN Operation**

* + If 2 relations R & S have at least one common attribute, then they can participate in a JOIN operation.
  + It is denoted by symbol .
  + The join operator allows combining of 2 relations to form a new relation.
  + The general form of a join operation on two relations R(A1, A2, . . ., An) and S(B1, B2, . . ., Bm) is:

**R <join condition>S**

where R and S can be any relations that result from general *relational algebra expressions.*

* + The join operation is a combination of the Cartesian product, selection and possibly projection operations.
  + If R & S are 2 relations, then join can operates as follows:

i) First it will form the Cartesian product of R & S i.e R X S

ii) Do a selection operation to eliminate some tuples (this operation is specified as

part of the join operation).

iii) Then optionally remove duplicate attributes with project operation.

## **Example 1: R S**

|  |  |
| --- | --- |
| NAME | ROLL |
| x | 101 |
| y | 102 |
| z | 301 |

|  |  |
| --- | --- |
| ROLL | GAME |
| 102 | cricket |
| 302 | chess |

|  |  |  |  |
| --- | --- | --- | --- |
| NAME | ROLL | ROLL | GAME |
| x | 101 | 102 | cricket |
| x | 101 | 302 | chess |
| y | 102 | 102 | cricket |
| y | 102 | 302 | chess |
| z | 301 | 102 | cricket |
| z | 301 | 302 | chess |

**R X S**

**JOIN OR EQUI JOIN OR INNER JOIN**

* The most common use of join involves join conditions with *equality comparisons* only
* Such a join, where the only comparison operator used is =, is called an EQUIJOIN

**Example 1:**

|  |  |  |  |
| --- | --- | --- | --- |
| NAME | ROLL | ROLL | GAME |
| y | 102 | 102 | cricket |

**R R.ROLL=S.ROLLS**

**Example 2:**

**(a) CUSTOMERS table is as follows:**

**+----+----------+-----+-----------+----------+**

**| ID | NAME | AGE | ADDRESS | SALARY |**

**+----+----------+-----+-----------+----------+**

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

**+----+----------+-----+-----------+----------+**

**(b) Another table is ORDERS as follows:**

**+-----+---------------------+-------------+--------+**

**|OID | DATE | CUSTOMER\_ID | AMOUNT |**

**+-----+---------------------+-------------+--------+**

**|** 102 | 2009-10-08 00:00:00 | 3 | 3000 |

| 100 | 2009-10-08 00:00:00 | 3 | 1500 |

| 101 | 2009-11-20 00:00:00 | 2 | 1560 |

| 103 | 2008-05-20 00:00:00 | 4 | 2060 |

SQL> SELECT ID, NAME, AMOUNT, DATE FROM CUSTOMERS, ORDERS WHERE

CUSTOMERS.ID = ORDERS.CUSTOMER\_ID;

Equivalent **relational algebra** is:

TEMP1🡨CUSTOMERS CUSTOMERS.ID = ORDERS.CUSTOMER\_ID ORDERS

**Π** ID, NAME, AMOUNT, DATE  (TEMP1)

This would produce the following result:

+----+----------+--------+---------------------+

| ID | NAME | AMOUNT | DATE |

+----+----------+--------+---------------------+

| 3 | kaushik | 3000 | 2009-10-08 00:00:00 |

| 3 | kaushik | 1500 | 2009-10-08 00:00:00 |

| 2 | Khilan | 1560 | 2009-11-20 00:00:00 |

| 4 | Chaitali | 2060 | 2008-05-20 00:00:00 |

+----+----------+--------+---------------------+

**NATURAL JOIN Operation**

* Another variation of JOIN called NATURAL JOIN — denoted by \*.
* It is a binary operation and a combination of certain selections and a Cartesian product into one operation.
* It forms a Cartesian product of its two arguments, then performs a selection forcing equality on those attributes those appear in both the relations. And finally removes duplicates attributes.

|  |  |  |
| --- | --- | --- |
| NAME | ROLL | GAME |
| y | 102 | cricket |

**Example 1:** **RESULT= R \* S**

**Example 2:** To apply a natural join on the CUSTOMER ID attributes of CUSTOMERS and

ORDERS

**RESULT ← CUSTOMERS \* ORDERS**

**The OUTER JOIN Operation**

* In NATURAL JOIN tuples without a *matching* (or *related*) tuple are eliminated from the join result. Tuples with null in the join attributes are also eliminated. This amounts to loss of information.
* A set of operations, called **outer joins**, can be used when we want to keep all the tuples in R, or all those in S, or all those in both relations in the result of the join, regardless of whether or not they have matching tuples in the other relation.

i) The ***left outer join*** operation keeps every tuple in the *first* or *left* relation R in R

S; if no matching tuple is found in S, then the attributes of S in the join

result are filled or “padded” with null values.

**SQL> SELECT ID, NAME, AMOUNT, DATE FROM CUSTOMERS LEFT JOIN ORDERS ON CUSTOMERS.ID = ORDERS.CUSTOMER\_ID;**

Equivalent relational algebra is:

**TEMP1=CUSTOMER ORDERS**

**RESULT = Π ID, NAME, AMOUNT, DATE (TEMP1)**

This would produce the following result:

+----+----------+--------+---------------------+

| ID | NAME | AMOUNT | DATE |

+----+----------+--------+---------------------+

| 1 | Ramesh | NULL | NULL |

| 2 | Khilan | 1560 | 2009-11-20 00:00:00 |

| 3 | kaushik | 3000 | 2009-10-08 00:00:00 |

| 3 | kaushik | 1500 | 2009-10-08 00:00:00 |

| 4 | Chaitali | 2060 | 2008-05-20 00:00:00 |

| 5 | Hardik | NULL | NULL |

| 6 | Komal | NULL | NULL |

| 7 | Muffy | NULL | NULL |

+----+----------+--------+---------------------+

ii) A similar operation, ***right outer join***, keeps every tuple in the *second* or right

relation S in the result of R S.

**SQL> SELECT ID, NAME, AMOUNT, DATE FROM CUSTOMERS RIGHT JOIN ORDERS ON CUSTOMERS.ID = ORDERS.CUSTOMER\_ID;**

Equivalent **relational algebra** is:

**TEMP1=CUSTOMER ORDERS**

**RESULT = Π** ID, NAME, AMOUNT, DATE (TEMP1)

This would produce the following result:

+------+----------+--------+---------------------+

| ID | NAME | AMOUNT | DATE |

+------+----------+--------+---------------------+

| 3 | kaushik | 3000 | 2009-10-08 00:00:00 |

| 3 | kaushik | 1500 | 2009-10-08 00:00:00 |

| 2 | Khilan | 1560 | 2009-11-20 00:00:00 |

| 4 | Chaitali | 2060 | 2008-05-20 00:00:00 |

+------+----------+--------+---------------------+

***iii)*** A third operation, ***full outer join***, denoted by R S keeps all tuples in

both the left and the right relations when no matching tuples are found, padding them

with null values as needed.

**SQL> SELECT ID, NAME, AMOUNT, DATE FROM CUSTOMERS FULL JOIN ORDERS ON CUSTOMERS.ID = ORDERS.CUSTOMER\_ID;**

Equivalent **relational algebra** is:

**TEMP1=CUSTOMER ORDERS**

**RESULT = Π** ID, NAME, AMOUNT, DATE  (TEMP1)

This would produce the following result:

+------+----------+--------+---------------------+

| ID | NAME | AMOUNT | DATE |

+------+----------+--------+---------------------+

| 1 | Ramesh | NULL | NULL |

| 2 | Khilan | 1560 | 2009-11-20 00:00:00 |

| 3 | kaushik | 3000 | 2009-10-08 00:00:00 |

| 3 | kaushik | 1500 | 2009-10-08 00:00:00 |

| 4 | Chaitali | 2060 | 2008-05-20 00:00:00 |

| 5 | Hardik | NULL | NULL |

| 6 | Komal | NULL | NULL |

| 7 | Muffy | NULL | NULL |

| 3 | kaushik | 3000 | 2009-10-08 00:00:00 |

| 3 | kaushik | 1500 | 2009-10-08 00:00:00 |

| 2 | Khilan | 1560 | 2009-11-20 00:00:00 |

| 4 | Chaitali | 2060 | 2008-05-20 00:00:00 |

**DIVISION Operation**

* It is a binary operation which is denoted by **÷.**
* This operation produces a relation that consists of the set of tuples from R defined over the attributes C that matche the combination of every tuple in S, where C is the set of attributes that are in R but not in S.
* This symbol is used for select “for all”.

**R S** **Result = R ÷ S : Here all a1, a5 are common to (b1, b2)**

|  |  |
| --- | --- |
| A | B |
| a1 | b1 |
| a1 | b2 |
| a2 | b1 |
| a3 | b1 |
| a4 | b2 |
| a5 | b1 |
| a5 | b2 |

|  |
| --- |
| A |
| a1 |
| a5 |

|  |
| --- |
| B |
| b1 |
| b2 |

**Summary of Relational Algebra Operations**

**
Table0601.tif                                                  0000732BEeyore                         BC0F1D1E:**

**Examples of Queries in Relational Algebra**

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

RESEARCH\_DEPT ← **σ** DNAME=’Research’ (DEPARTMENT)

RESEARCH\_EMPS ← (RESEARCH\_DEPT DNUMBER= DNOEMPLOYEEEMPLOYEE)

RESULT ←π FNAME, LNAME, ADDRESS (RESEARCH\_EMPS)

**Q2: Retrieve the names of employees who have no dependents.**

ALL\_EMPS ← π SSN(EMPLOYEE)

EMPS\_WITH\_DEPS(SSN) ← π ESSN(DEPENDENT)

EMPS\_WITHOUT\_DEPS ← (ALL\_EMPS - EMPS\_WITH\_DEPS)

RESULT ← π LNAME, FNAME (EMPS\_WITHOUT\_DEPS \* EMPLOYEE)

Do more examples