RELATIONAL DATA MODEL

3.1 Introduction

The relational model of data was introduced by Codd (1970). It is based on a simple and uniform data structure - the relation - and has a solid theoretical and mathematical foundation.

The relational model is becoming firmly established in the database application world, and there are many commercial relational DBMSs.

The first relational products began to appear in the late 1970s and early 1980s. At present time, many relational DBMSs are commercially available, and those products run on just about every kind of hardware and software platform imaginable. Just few examples of relational DBMSs are:

- DB2, SQL/DS, the OS/2 Extended Edition Database Manager, and OS/400 Database Manager (all from IBM).
- Rdb/VMS from Digital Equipment Corporation.
- ORACLE from Oracle Corporation.
- MS-ACCESS from Microsoft.

More recently, research has proceeded on a variety of might be called “postrelational” systems, some of them based on upward-compatible extensions to the original relational approach, others representing attempts at doing something entirely different. The domain of these researches includes:

- Deductive DBMSs.
- Semantic DBMSs.
- Universal relation DBMSs.
- Object-oriented DBMSs.
- Extendable DBMSs.
- Expert DBMSs.

3.2 Relational Model Concepts
The relational model represents the database as a collection of relations. Informally, each relation resembles a table or, to some extent, a simple file. When a relation is thought of as a table of values, each row in the table represents a collection of related data values. These values can be interpreted as facts describing a real world entity or relationship. The table name and column names are used to help in interpreting the meaning of the values in each row of the table.

For example, consider the suppliers-and-parts database in Figure 3.1. The first table in that database is called SUPPLIER because each row represents facts about a particular supplier entity. The column names - SNO, Sname, Status, City - specify how to interpret the data values in each row, based on the column each value is in. All values in a column are of the same data type.

**SUPPLIER**

<table>
<thead>
<tr>
<th>SNO</th>
<th>Sname</th>
<th>Status</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>Ahmed</td>
<td>20</td>
<td>Cairo</td>
</tr>
<tr>
<td>S2</td>
<td>Badran</td>
<td>10</td>
<td>Cairo</td>
</tr>
<tr>
<td>S3</td>
<td>Aly</td>
<td>10</td>
<td>Alex</td>
</tr>
<tr>
<td>S4</td>
<td>Sadek</td>
<td>20</td>
<td>Cairo</td>
</tr>
</tbody>
</table>

**PART**

<table>
<thead>
<tr>
<th>PNO</th>
<th>Pname</th>
<th>Color</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Nut</td>
<td>Red</td>
<td>12</td>
</tr>
<tr>
<td>P2</td>
<td>Bolt</td>
<td>Green</td>
<td>15</td>
</tr>
<tr>
<td>P3</td>
<td>Screw</td>
<td>Blue</td>
<td>15</td>
</tr>
<tr>
<td>P4</td>
<td>Cam</td>
<td>Red</td>
<td>17</td>
</tr>
<tr>
<td>P5</td>
<td>Screw</td>
<td>Black</td>
<td>14</td>
</tr>
</tbody>
</table>

**SUPPLY**

<table>
<thead>
<tr>
<th>SNO</th>
<th>PNO</th>
<th>QTY</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>P1</td>
<td>100</td>
</tr>
<tr>
<td>S1</td>
<td>P2</td>
<td>200</td>
</tr>
<tr>
<td>S1</td>
<td>P3</td>
<td>100</td>
</tr>
<tr>
<td>S2</td>
<td>P1</td>
<td>150</td>
</tr>
<tr>
<td>S2</td>
<td>P3</td>
<td>100</td>
</tr>
<tr>
<td>S3</td>
<td>P2</td>
<td>200</td>
</tr>
<tr>
<td>S4</td>
<td>P3</td>
<td>300</td>
</tr>
<tr>
<td>S4</td>
<td>P2</td>
<td>100</td>
</tr>
</tbody>
</table>

Figure 3.1 The suppliers-and-parts relational database
3.3 Relational Data Structure

In relational model terminology, the terms in question are (Figure 3.2):

- relation,
- tuple,
- attribute,
- cardinality,
- degree,
- domain, and,
- primary key.

![EMPLOYEE relation](image)

- A relation corresponds to a table.
- A tuple corresponds to a row of such a table.
- An attribute corresponds to a column.
- The cardinality is the number of tuples. For example, the cardinality of relation EMPLOYEE is 4, while the cardinality of the relation SUPPLY (Figure 3.1) is 8.
- The degree is the number of attributes. For example, the degree of relation EMPLOYEE is 5, while the degree of the relation SUPPLY is 3.
- The primary key is a unique identifier for the table - that is an attribute or attribute combination with the property of uniqueness, that is, at any given time, no two rows of the table contain the same value in that attribute or
attribute combination. For example, the primary key of relation EMPLOYEE is selected to be the attribute E#.

- A domain is a pool of values, from which one or more attributes (columns) draw their actual values. A domain is a set of atomic values. By atomic we mean that each value in the domain is indivisible as far as the relational model is concerned. A common method of specifying a domain is to specify a data type from which the data values forming the domain are drawn. It is also useful to specify a name for the domain, to help in interpreting its values. Some examples of domains that can be defined for relation EMPLOYEE are:
  - Emp_Id: The set of ID numbers of employees.
  - Names: The set of names of persons.
  - Emp_Sex: The set of sex values of persons. In this case, Emp_Sex = {M, F}.
  - Emp_Salary: The range of salaries of employees. For example, Emp_Salary can be defined as:
    - Emp_Salary = {salary | 1000 ≤ salary ≤ 3000}

- A data type is also specified for each domain. For example, the data type for the domain Emp_Id can be declared as an integer, while the data type of the domain Names is a character string of length 40, data type of the domain Emp_Salary is numeric, and so on.

- A format can be specified for each domain. For example, the domain Phones can be declared as a character string of the form (ddd) ddd-dddd, where each d is a numeric (decimal) digit and the first three digits form a valid telephone area code.

### 3.3.1 Relation schema

A relation schema R, denoted by R(A₁, A₂, ..., Aₙ), is made up of a relation name R and a list of attributes A₁, A₂, ..., Aₙ. Each attribute Aᵢ is defined on a domain D. The domain D is called the domain of Aᵢ and is denoted by dom(Aᵢ). A relation schema is used to describe a relation; R is called the name of this relation. The degree of a relation is the number of attributes n of its relation schema. An example of a relation schema of degree 5, which describes an employee in a company, is the following (Figure 3.2):

EMPLOYEE(E#, Name, Phone, Sex, Salary)
For this relation schema, EMPLOYEE is the name of the relation, which has five attributes. We can specify the following domains for some attributes of the EMPLOYEE relation:

- \( \text{dom(E\#)} = \text{Emp\_Id} \)
- \( \text{dom(\text{Name})} = \text{Names} \)
- \( \text{dom(\text{Phone})} = \text{Phones} \)
- \( \text{dom(\text{Sex})} = \text{Emp\_sex} \)
- \( \text{dom(\text{Salary})} = \text{Emp\_salary} \)

### 3.3.2 Relation instance

A relation (or relation instance) \( r \) of the relational schema \( R(A_1, A_2, \ldots, A_n) \), also denoted by \( r(R) \), is the set of \( n \)-tuples \( r = \{t_1, t_2, \ldots, t_m\} \). Each \( n \)-tuple \( t \) is an ordered list of \( n \) values \( t = <v_1, v_2, \ldots, v_n> \), where each value \( v_i \), \( 1 \leq i \leq n \), is an element of \( \text{dom}(A_i) \) or is a special null value. The terms relation intension for the schema \( R \) and relation extension (or state) for a relation instance \( r(R) \) are also commonly used.

Figure 3.2 shows an example of an EMPLOYEE relation, which corresponds to the EMPLOYEE schema specified above. Each tuple in the relation represents a particular employee entity. We represent the relation as a table, where each tuple is shown as a row and each attribute corresponds to a column. Null values represents attributes whose values are unknown or do not exist for some individual EMPLOYEE tuples.

A relation instance at a given time - the current relation state - reflects only valid tuples that represent a particular state of the real world. In general, as the state of the real world changes, so does the relation instance, by being transformed into another relation instance. However, the schema \( R \) is relatively static and does not change except very infrequently - for example, as a result of adding an attribute to represent new information that was not originally stored in the relation.

### 3.3.3 Properties of Relations

Relations possess certain properties, all of them are immediate consequences of the definition of the relation schema and instance. These important properties are:

- **There are no duplicate tuples.** This property follows from the fact that the relation instance is a mathematical set (i.e., a set of tuples), and sets
mathematics by definition do not include duplicate elements. An important corollary of the fact that there are no duplicate tuples is that **there is always a primary key**. Since tuples are unique, it follows that at least the combination of all attributes of the relation schema has the uniqueness property, so that at least the combination of all attributes can (if necessary) serves as the primary key. Incidentally, this first property serves right away as an illustration of the point that a relation and a table are not the same thing, because a table (in general) might contain duplicate rows whereas a relation cannot contain any duplicate tuples.

- **Tuples are unordered (top to down).** This property also follows from the fact that the relation instance is a mathematical set. Sets in mathematics are not ordered. In Figure 3.2, for example, the tuples of relation EMPLOYEE could just as well have been shown in the reverse sequence - it would still have been the same relation. Thus, there is no such thing as “the 5th tuple” or “the next tuple”; in other words, there is no concept of positional addressing. This property also serves to illustrates the point that a relation and a table are not the same thing, because the rows of a table obviously do have a top-to-bottom ordering, whereas the tuples of relation do not.

- **Attributes are unordered (left to right).** This property also follows from the fact that the relation schema is a mathematical set of attributes. Sets in mathematics are not ordered. In Figure 3.2, for example, the attributes of relation EMPLOYEE could just as well have been shown in the order (say) Name, Phone, Sex, E#, Salary - it would still have been the same relation, at least so far as the relational model is concerned. Thus, there is no such thing as “the 1th attribute” or “the next attribute”. In the case of relation EMPLOYEE, for example, there is no “3rd attribute”; instead, there is a Sex attribute, which is always referenced by name, never by position. This question of attribute ordering is yet another area where the concrete representation of a relation as a table suggests something that is not really true: The columns of a table obviously do have a left-to-right ordering, but the attributes of a relation do not.

- **All attribute values are atomic.** This property is a consequence of the fact that all underlying domains are atomic (simple), i.e., contain atomic values only. This property means that at every row-and-column position within the table, there always exists precisely one value, never a list of values. Or equivalently: Relations do not contain repeating groups. A relation satisfying this condition is said to be **normalized**. The foregoing implies that all relations are normalized so far as the relational model is concerned.
The values of some attributes within a particular tuple may be unknown or may not apply to that tuple. A special value, called null, is used for these cases. For example, in Figure 3.2, some employee tuples have null for their phones because they do not have phones in their offices.

### 3.4 Relational Integrity Constraints

The problem of integrity is the problem of ensuring that the data in the database is accurate and consistent.

Integrity of a database can be maintained by enforcing a set of constraints that should hold on the data stored in the database. These constraints are of two types:

- **Specific integrity constraints** which are all the constraints that should hold for a particular database. For example, in the database of suppliers-and-parts given in Figure 3.1, the following constraints may be enforced:
  - Part weights must be greater than zero (weights cannot be negative in the real world);
  - Supplier status values must be in range 1-100;
  - Part colors must be drawn from a certain list;
  - Supplier cities must be drawn from a certain list;
  - Supply quantities must be greater than zero;

However, all these rules are specific, in the sense that they apply to specific database.

- General integrity constraints. These constraints are general, in the sense that they apply, not just to some specific database such as suppliers-and-parts, but rather to every relational database. There are two general integrity constraints (or rules):
  1. **The entity integrity constraint**, and
  2. **The referential constraint**.

These two integrity constraints have to do with *primary keys* and *foreign keys*.

### 3.4.1 Primary keys and Candidate Keys

The primary key is a special case of a general key called the *candidate key*. Informally, the candidate key of a relation is just a unique identifier for that relation. Every relation has at least one candidate key (relations do not
contain duplicate tuples). In practice, most relations tend to have exactly one candidate key, but it is certainly possible to have more than one.

If a relation has only one candidate key, then this candidate key is named the primary key for that relation. But, if a relation has more than one candidate key, then, exactly one is chosen as the primary key for that relation; the remainder, if any, are called alternate keys.

For example, the candidate keys for relations SUPPLIER, PART, and SUPPLY of the suppliers-and-parts database are SUPPLIER.SNO, PART.PNO, and SUPPLY.(SNO,PNO), respectively. Hence, SUPPLIER.SNO, PART.PNO, and SUPPLY.(SNO, PNO) are the primary keys for relations SUPPLIER, PART, and SUPPLY, respectively.

If we suppose that every supplier always has a unique supplier number (SNO) and a unique supplier name (Sname). In such a case we would say that the relation SUPPLIER has two candidate keys, namely, SNO and Sname. We would then choose one of those candidate keys (say SNO) to be the primary key, and the remainder (Sname) would then be said to be alternate key.

Now, we come to a basic question: What are the conditions that should be satisfied in any attribute or attributes combination such that it can be a candidate for the underlying relation?

**Candidate Key**

Attribute \( K \) (possibly composite) of relation \( R \) is a candidate key for relation \( R \) if and only if it satisfies the following two time-independent properties:

1. **Uniqueness**: At any given time, no two tuples of \( R \) have the same value for \( K \).
2. **Minimality**: If \( K \) is composite, then no component of \( K \) can be eliminated without destroying the uniqueness property.

For example, in relation SUPPLIER, it is assumed that SNO is unique, meaning that no two tuples in relation SUPPLIER may have the same value for SNO. Hence, SNO is a candidate key for relation SUPPLIER (because it satisfies uniqueness and minimality properties), and by consequence it is the primary key for that relation. By contrast, the composite attribute (SNO, City) is not a candidate key for relation SUPPLIER, because, though it does satisfy the uniqueness property, but it does not satisfy the minimality property (City is irrelevant for unique identification - it can be eliminated without destroying the uniqueness property).
Primary Key

The primary key for relation R is selected from the set of candidate keys of that relation (as explained above).

Since every relation has at least one candidate key, it follows that every relation has a primary key.

It is important to understand that, in practice, it is the primary key that is really significant one; candidate and alternate keys are merely concepts that necessarily arise during the process of defining the more important concept “primary key”.

Why are primary keys important? A fundamental answer is that primary keys provide the basic tuple-level addressing mechanism in a relational system. That is, the only system-guaranteed way of locating and selecting some specific tuple is by its primary key.

3.4.2 The Entity Integrity Constraint

Now we come to the first of the two general integrity constraints of the relational model, namely the entity integrity constraint. The constraint is very simple, and runs as follows:

◊ No component of the primary key of a base relation is allowed to accept nulls.

Null is simply a value or representation that is understood by convention not to stand for any real value of the applicable attribute.

The justification for the entity integrity constraint is as follows:

• Base relations (or, more precisely, tuples within base relations) correspond to entities in the real world, For example, base relation SUPPLIER corresponds to a set of suppliers in the real world.

• By definition, entities in the real world are distinguishable - that is, they are identifiable in some way.
• Therefore, entity representatives within the database must be distinguishable (identifiable) also.

• Primary keys perform this unique identification function in the relational model (i.e., they serve to represent the necessary entity identifiers).

• Suppose, therefore, by way of example, that base relation SUPPLIER included a tuple for which SNO value was null. Then that would be like saying that there was a supplier in the real world that has no identity.

• If that null means “property does not apply”, then clearly the tuple makes no sense; as explained above, entities must have identity, and hence the property must apply.

• If it means “value is unknown”, then all kinds of problems arise. For example, we now do not even know (in general) whether the tuple represents one of the suppliers we do know about.

• To sum up: If an entity is important enough in the real world to require explicit representation in the database, then that entity must be definitely and unambiguously identifiable - for otherwise it would be impossible even to talk about it in any sensible manner. For this reason, the entity integrity constraint is sometimes stated in the form:

   “In a relational database, we never record information about something we cannot identify”.

Important points:

1- In the case of composite primary key, the entity integrity constraint says that every individual value of the primary key must be nonnull.

2- The entity integrity constraint applies to base relations. Other relations, such as output relations of certain queries, might very well have a primary key for which nulls are allowed. As a trivial example, suppose that nulls are allowed for attribute PART.Color in the suppliers-and-parts database, and consider the relation that results from the query “List all part colors”.

(10)
3- Entity integrity constraint applies only to primary key. Alternate keys may or may not have “nulls allowed”.

3.4.3 The Referential Integrity Constraint

The referential integrity constraint is the second general integrity constraint in the relational model. This integrity constraint has to do with primary keys and foreign keys.
Refer once again to the suppliers-and-parts database, and consider attribute SNO of relation SUPPLY. It is clear that a given value for that attribute should be permitted to appear in the database only if that same value also appears as a value of the primary key SNO of relation SUPPLIER (for otherwise the database cannot be considered to be in state of integrity). For example, it would make no sense for relation SUPPLY to include a SUPPLY for supplier S9 (say) if there were no supplier S9 in relation SUPPLIER. Likewise, a given value for attribute PNO of relation SUPPLY should be permitted to appear only if the same value also appears as a value of the primary key PNO of relation PART; for again it would make no sense for relation SUPPLY to include a SUPPLY for part P8 (say) if there were no part P8 in relation PART.

Attributes SNO and PNO of relation SUPPLY are examples of what are called foreign keys.

Generally, a Foreign Key is an attribute (possible composite) of one relation R2 whose values are required to match those of the primary key of some relation R1 (R1 and R2 not necessarily distinct).

As a formal definition of the term “foreign key”, attribute FK (possibly composite) of base relation R2 is a foreign key if and only if it satisfies the following two time-independent properties.

1- Each value of FK is either wholly null or wholly nonnull. (By “wholly null or wholly nonnull,” we mean that, if FK is composite, then each value of FK either has all components null or all components nonnull, not a mixture.)

2- There exists a base relation R1 with primary key PK such that each nonnull value of FK is identical to the value of PK in some tuple of R1.

Concerning that definition of foreign key, the following points should be considered:

1- A given foreign key and the corresponding primary key should be defined on the same domain.

2- There is no requirement that a foreign key be a component of the primary key of its containing relation, although in the case of suppliers-and-parts it does so happen that both foreign keys in fact are. Here is a counterexample (departments and employees):
In this database, attribute EMP.DEPTNO is a foreign key in relation EMP (matching the primary key DEPT.DEPTNO of relation DEPT); however, it is not a component of the primary key EMP.EMPNO of that relation EMP. In general, any attribute whatsoever (in a base relation) can be a foreign key.

3- Relations R1 and R2 in the foreign key definition are not necessarily distinct. That is, a relation might include a foreign key whose (nonnull) values are required to match the values of the primary key of that same relation. For example, consider the relation:

EMP ( EMPNO, ..., SUPERVISOR_EMPNO ... )

In that relation attribute SUPERVISOR_EMPNO represents the employee number of the supervisor of the employee identified by EMPNO. Here EMPNO is the primary key, and SUPERVISOR_EMPNO is a foreign key that refers to it.

4- The foreign keys, unlike primary keys (in base relations), do sometimes have to accept nulls. As an example, consider again the relation EMP discussed in paragraph 3 above. What is the value of SUPERVISOR_EMPNO for the president of the company?

5- Foreign-to-primary-key matches are sometimes said to be the “glue” that holds the database together. In other words, the foreign-to-primary-key matches represent certain relationships between tuples.

The Referential Integrity Constraint

We can now state the second general integrity constraint of the relational model, the referential integrity constraint.

◊ The database must not contain any unmatched foreign key values.

By the term “unmatched foreign key value” we mean a nonnull foreign key value for which there does not exist a matching value of the primary key in the relevant target relation.
The justification of this constraint is surely obvious: Just as primary key values represent entity identifiers, so foreign key values represent entity references (unless, of course, they happen to be null). The referential integrity constraint simply says that if $B$ references $A$, then $A$ must exist.

### 3.5 ER-to-Relational Mapping

The relational schema for a particular database can be obtained from the ER schema of that database by applying a well-defined mapping procedure. As an example, that procedure is applied on the ER schema for the COMPANY database (Figure 3.3).
Figure 3.3 ER schema diagram for the company database
3.5.1 ER-to-Relational Mapping Procedure

- **STEP 1**: For each regular entity type $E$ in the ER schema, we create a relation $R$ that includes all the simple attributes of $E$. For a composite attribute we include only the simple component attributes. We choose one of the key attributes of $E$ as primary key for $R$. If the chosen key of $E$ is composite, then the set of simple attributes that form it will together form the primary key of $R$. In our example we create the relations EMPLOYEE, DEPARTMENT, and PROJECT in Figure 3.4 to correspond to the regular entity types EMPLOYEE, DEPARTMENT, and PROJECT. We choose SSN, DNUMBER, and PNUMBER as primary keys for the relations EMPLOYEE, DEPARTMENT, and PROJECT respectively.

- **STEP 2**: For each weak entity type $W$ in the ER schema with owner entity type $E$, we create a relation $R$ and include all simple attributes (or simple components of composite attributes) of $W$ as attributes of $R$. In addition, we include as foreign key attributes of $R$ the primary key attribute(s) of the relation that corresponds to the owner entity type $E$; this takes care of the identifying relationship type of $W$. The primary key of $R$ is the combination of the primary key of the owner and the partial key of the weak entity type $W$. In our example we create the relation DEPENDENT in this step to correspond to the weak entity type DEPENDENT. We include the primary key of the EMPLOYEE relation - which corresponds to the owner entity type - as an attribute of DEPENDENT; we renamed it to ESSN although this is not necessary.
Figure 3.4  Relational database schema corresponding to the ER schema of Figure 3.3
The primary key of the DEPENDENT relation is the combination (ESSN, DEPENDENT_NAME) because DEPENDENT_NAME is the partial key of DEPENDENT.

- **STEP 3**: For each binary 1:1 relationship type R in the ER schema, we identify the relations S and T that correspond to the entity types participating in R. We choose one of the relations, S say, and include as 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. We include all the simple attributes (or simple components of a composite attribute) of the 1:1 relationship type R as attributes of S. In our example we map the 1:1 relationship type MANAGES from Figure 3.3 by choosing the participating entity type DEPARTMENT to serve in the role of S because its participation in the MANAGES relationship type is total (every department has a manager). We include the primary key of the EMPLOYEE relation as foreign key in the DEPARTMENT relation and rename it MGRSSN. We also include the simple attribute StartDate of the MANAGES relationship type in the DEPARTMENT relation and rename it MGRSTARTDATE.

- **STEP 4**: For each regular (nonweak) binary 1:N relationship type R, we identify the relation S that represents the participating entity type that participates once in R. We include as a foreign key in S the primary key of relation T that represents the other entity type participating in R. This is because each entity instance of the entity type participating once in relationship type R is related to at most one entity instance from the other entity type. For example, in the 1:N relationship type WORKS_FOR, each employee is related to one department. We include any simple attributes (or simple components of composite attributes) of the 1:N relationship type as attributes of S. In our example we map the 1:N relationship type WORKS_FOR and SUPERVISION from Figure 3.3. For WORKS_FOR we include the primary key of the DEPARTMENT relation as foreign key in the EMPLOYEE relation and call it DNO. For SUPERVISION we include the primary key of the EMPLOYEE relation as foreign key in the EMPLOYEE relation itself and call it SUPERSSN. The CONTROLS relationship type is similarly mapped.

- **STEP 5**: For each binary M:N relationship type R, we create a new relation S to represent R. We include as foreign key attributes in R the primary keys of the relations that represent the participating entity types; their combination will form the primary key of S. We include any simple attributes of the M:N relationship type (or simple components of composite attributes) as attributes
of S. Notice that we cannot represent an M:N relationship type by a single foreign key attribute in one of the participating relations - as we did for 1:1 and 1:N relationship types - because of the M:N cardinality ratio. In our example, we map the M:N relationship type WORKS_ON from Figure 3.3 by creating the relation WORKS_ON in Figure 3.4. We include the primary keys of the EMPLOYEE and PROJECT relations as foreign keys in WORKS_ON and rename them ESSN and PNO respectively. We also include an attribute HOURS in WORKS_ON to represent the Hours attribute of the relationship type. The primary key of WORKS_ON relation is the combination of the foreign key attributes (ESSN, PNO).

**STEP 6:** For each multivalued attribute A, we create a new relation R that includes an attribute corresponding to A plus the primary key attribute K of the relation that represents the entity type or 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. In our example, we create a relation DEPT_LOCATIONS. The attribute DLOCATION represents the multivalued attribute Location of DEPARTMENT, while DNUMBER - as foreign key - represents the primary key of the DEPARTMENT relation. The primary key of DEPT_LOCATIONS is the combination of (DNUMBER, DLOCATION). A separate tuple will exist in DEPT_LOCATIONS for each location that a department has.

Figure 3.4 shows the relational schema obtained by the above steps. Notice that we didn’t discuss the mapping of n-ary relationship type (n > 2) because none exist in Figure 3.3; these can be mapped in a similar way to M:N relationship types by including the following additional step in the mapping procedure.

**STEP 7:** For each n-ary relationship type R (n > 2), we create a new relation S to represent R. We include as foreign key attributes in S the primary keys of relations that represent the participating entity types. We 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 the combination of all the foreign keys that reference the relations representing the participating entity types. For example, consider the relationship type SUPPLY of Figure 3.5; this can be mapped to relation SUPPLY shown in Figure 3.6 whose primary key is the combination of foreign keys (SNAME, PARTNO, PROJNAME).

*This concludes the mapping procedure*
Figure 3.5 The Ternary relationship type SUPPLY with n = 3

SUPPLIER

<table>
<thead>
<tr>
<th>SNAME</th>
</tr>
</thead>
</table>

PROJECT

<table>
<thead>
<tr>
<th>PROJNAME</th>
</tr>
</thead>
</table>

PART

<table>
<thead>
<tr>
<th>PARTNO</th>
</tr>
</thead>
</table>

SUPPLY

<table>
<thead>
<tr>
<th>SNAME</th>
<th>PROJNAME</th>
<th>PARTNO</th>
<th>QUANTITY</th>
</tr>
</thead>
</table>

Figure 3.6 Corresponding relational schema.
3.5.2 ER schema versus Relational schema

The main point to note in a relational schema as compared to an ER schema is that relationship types are not represented explicitly; they are represented by having two attributes A and B, one a primary key and the other a foreign key - over the same domain - included in two relations S and T. Two tuples in S and T are related when they have the same value for A and B. By using the EQUIJOIN (or NATURALJOIN) operation over S.A and T.B, we can combine all pairs of related tuples from S and T and materialize the relationship. When a binary 1:1 or 1:N relationship type is involved, a single join operation is usually needed. For a binary M:N relationship type, two join operations are needed, whereas for n-ary relationship types, n joins are needed. For example, to form a relation that includes the employee name, project name, and hours that the employee works on each project, we need to connect each EMPLOYEE tuple to the related PROJECT tuples via the WORKS_ON relation. Hence, we must apply the EQUIJOIN operation to the EMPLOYEE and WORKS_ON relations with the join condition SSN = ESSN, and then apply another EQUIJOIN operation to the resulting relation and the PROJECT relation with the join condition PNO = PNUMBER. Figure 3.7 shows the pairs of attributes that are used in EQUIJOIN operations to materialize each relationship type in the COMPANY schema of Figure 3.3.

<table>
<thead>
<tr>
<th>ER Relationship</th>
<th>Participating Relations</th>
<th>Join Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORKS_FOR</td>
<td>EMPLOYEE, DEPARTMENT</td>
<td>EMPLOYEE.DNO = DEPARTMENT.DNUMBER</td>
</tr>
<tr>
<td>MANAGES</td>
<td>EMPLOYEE, DEPARTMENT</td>
<td>EMPLOYEE.SSN = DEPARTMENT.MGRSSN</td>
</tr>
<tr>
<td>SUPERVISION</td>
<td>EMPLOYEE(E), EMPLOYEE(S)</td>
<td>EMPLOYEE(E).SUPERSSN = EMPLOYEE(S).SSN</td>
</tr>
<tr>
<td>WORKS_ON</td>
<td>EMPLOYEE, WORKS_ON, PROJECT</td>
<td>EMPLOYEE.SSN = WORKS_ON.ESSN AND PROJECT.PNUMBER = WORKS_ON.PNO</td>
</tr>
<tr>
<td>CONTROLS</td>
<td>DEPARTMENT, PROJECT</td>
<td>DEPARTMENT.DNUMBER = PROJECT.DNUMBER</td>
</tr>
<tr>
<td>DEPENDENTS_OF</td>
<td>EMPLOYEE, DEPENDENT</td>
<td>EMPLOYEE.SSN = DEPENDENT.ESSN</td>
</tr>
</tbody>
</table>

Figure 3.7 Join Conditions for materializing the relationship types of the COMPANY ER schema
Another point to note in the relational schema is that we create a separate relation for each multivalued attribute. For a particular entity with a set of values for the multivalued attribute, the key attribute value of the entity is repeated once for each value of the multivalued attribute in a separate tuple. This is because the basic relational model does not allow multiple values (or set of values) for an attribute in a single tuple.

The correspondences between ER and relational model are summarized in Figure 3.8.

<table>
<thead>
<tr>
<th>ER Model</th>
<th>Relational Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity type</td>
<td>“entity” relation</td>
</tr>
<tr>
<td>1:1 or 1:N relationship type</td>
<td>foreign key (or “relationship” relation)</td>
</tr>
<tr>
<td>M:N relationship type</td>
<td>“relationship” relation and two foreign keys</td>
</tr>
<tr>
<td>n-ary relationship type</td>
<td>“relationship” relation and n foreign keys</td>
</tr>
<tr>
<td>simple attribute</td>
<td>attribute</td>
</tr>
<tr>
<td>composite attribute</td>
<td>set of simple component attributes</td>
</tr>
<tr>
<td>multivalued attribute</td>
<td>relation and foreign key</td>
</tr>
<tr>
<td>value set</td>
<td>domain</td>
</tr>
<tr>
<td>key attribute</td>
<td>primary (or alternate) key</td>
</tr>
</tbody>
</table>

Figure 3.8 Correspondence between ER and relational models
KEY POINTS

- The relational model of data was introduced by Codd (1970). It is based on a simple and uniform data structure - the relation - and has a solid theoretical and mathematical foundation. The relational model is becoming firmly established in the database application world, and there are many commercial relational DBMSs.

- A database is a collection of related data, where data means recorded facts. A typical database represents some aspect of the real world and is used for specific purposes by one or more groups of users.

- A relation is a mathematical structure consists of a schema and an instance. The schema consists of a set of attributes based on a set of domains. A relation schema has a primary key which is an attribute (possibly composite) satisfying the two constraints of uniqueness and minimality. The instance of a relation consists of a set tuples whose values are drawn from the underlying domains.

- Four important characteristics differentiate relations from ordinary tables or simple files. These characteristics are: (1) Duplicate tuples are not allowed, (2) Tuples are unordered, (3) Attributes are unordered, and (4) All attribute values are atomic.

- The problem of integrity is the problem of ensuring that the data in the database is accurate and consistent. In the relational model, integrity of a database can be maintained by enforcing two general integrity constraints that should hold on the data stored in the database. These constraints are: the entity integrity constraint, and the referential constraint. These two integrity constraints have to do with the primary keys and the foreign keys.

- The relational schema for a particular database can be obtained from the ER schema of that database by executing a general well-defined mapping procedure.