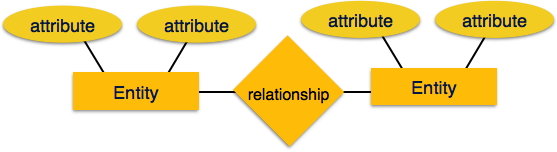
1. **ENTITY-RELATIONSHIP MODEL (ER Model)**

Entity-Relationship model is a high-level data model. Entity Relationship model is based on the notion of real world entities and relationship among them. It is pictorially represented as in Fig 1.14. ER Model is best used for the conceptual design of database. ER model is considered well for designing databases.

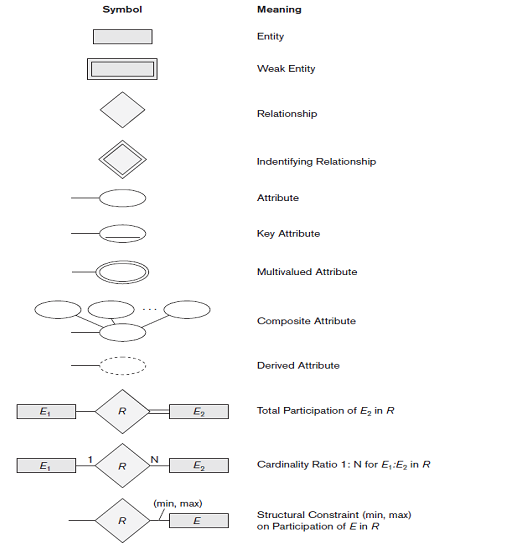
**ER Model is based on:**

1. **Entities and their *attributes and***
2. **Relationships among entities**

****

**Fig. 1.14. ER Model**

**The various symbols used in ER diagrams are depicted in Fig 1.15.**

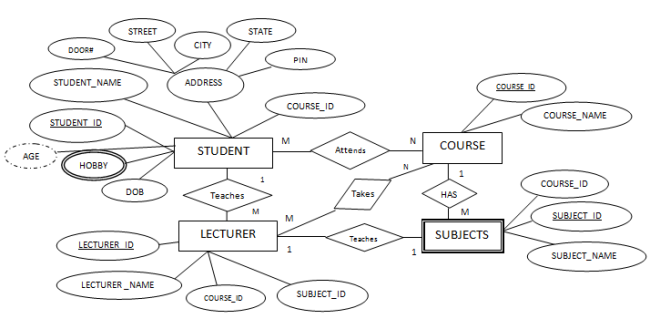
****

**Fig 1.15. Symbols used in ER Diagrams.**

**TRANSFORM ER DIAGRAM INTO TABLES**

Since ER diagram gives us the good knowledge about the requirement and the mapping of the entities in it, we can easily convert them as tables and columns. i.e.; using ER diagrams one can easily created relational data model, which nothing but the logical view of the database.

There are various steps involved in converting it into tables and columns. Each type of entity, attribute and relationship in the diagram takes their own depiction here. Consider the ER diagram below and will see how it is converted into tables, columns and mappings.



The basic rule for converting the ER diagrams into tables is

* Convert all the Entities in the diagram to tables.

All the entities represented in the rectangular box in the ER diagram become independent tables in the database. In the below diagram, STUDENT, COURSE, LECTURER and SUBJECTS forms individual tables.

* All single valued attributes of an entity is converted to a column of the table

All the attributes, whose value at any instance of time is unique, are considered as columns of that table. In the STUDENT Entity, STUDENT\_ID, STUDENT\_NAME form the columns of STUDENT table. Similarly, LECTURER\_ID, LECTURER\_NAME form the columns of LECTURER table. And so on.

* Key attribute in the ER diagram becomes the Primary key of the table.

In diagram above, STUDENT\_ID, LECTURER\_ID, COURSE\_ID and SUB\_ID are the key attributes of the entities. Hence we consider them as the primary keys of respective table.

* Declare the foreign key column, if applicable.

In the diagram, attribute COURSE\_ID in the STUDENT entity is from COURSE entity. Hence add COURSE\_ID in the STUDENT table and assign it foreign key constraint. COURSE\_ID and SUBJECT\_ID in LECTURER table forms the foreign key column. Hence by declaring the foreign key constraints, mapping between the tables are established.

* Any multi-valued attributes are converted into new table.

A hobby in the Student table is a multivalued attribute. Any student can have any number of hobbies. So we cannot represent multiple values in a single column of STUDENT table. We need to store it separately, so that we can store any number of hobbies, adding/ removing / deleting hobbies should not create any redundancy or anomalies in the system. Hence we create a separate table STUD\_HOBBY with STUDENT\_ID and HOBBY as its columns. We create a composite key using both the columns.

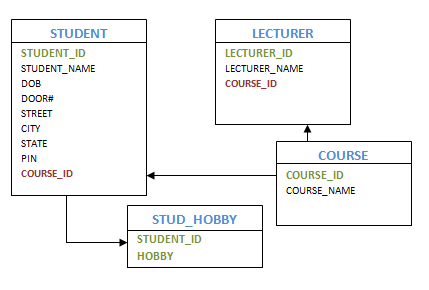
* Any composite attributes are merged into same table as different columns.

In the diagram above, Student Address is a composite attribute. It has Door#, Street, City, State and Pin. These attributes are merged into STUDENT table as individual columns.

* One can ignore derived attribute, since it can be calculated at any time.

In the STUDENT table, Age can be derived at any point of time by calculating the difference between DateOfBirth and current date. Hence we need not create a column for this attribute. It reduces the duplicity in the database.

These are the very basic rules of converting ER diagram into tables and columns, and assigning the mapping between the tables. Table structure at this would be as below:

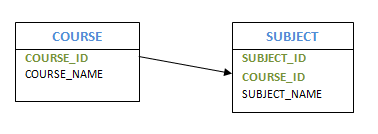


Let us see some of the special cases.

* **Converting Weak Entity**

Weak entity is also represented as table. All the attributes of the weak entity forms the column of the table. But the key attribute represented in the diagram cannot form the primary key of this table. We have to add a foreign key column, which would be the primary key column of its strong entity. This foreign key column along with its key attribute column forms the primary key of the table.

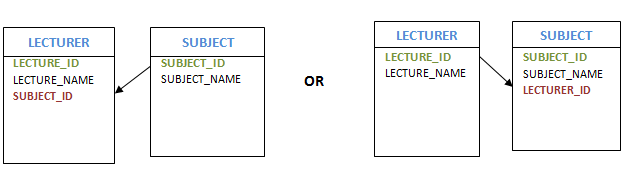
In our example above, SUBJECTS is the weak entity. Hence, we create a table for it. Its attributes SUBJECT\_ID and SUBJECT\_NAME forms the column of this table. Although SUBJECT\_ID is represented as key attribute in the diagram, it cannot be considered as primary key. In order to add primary key to the column, we have to find the foreign key first. COURSE is the strong entity related to SUBJECT. Hence the primary key COURSE\_ID of COURSE is added to SUBJECT table as foreign key. Now we can create a composite primary key out of COURSE\_ID and SUBJECT\_ID.



* **Representing 1:1 relationship**

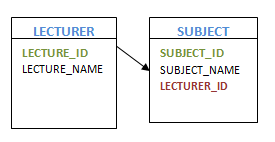
Imagine SUBJECT is not a weak entity, and we have LECTURER teaches SUBJECT relation. It is a 1:1 relation. i.e.; one lecturer teaches only one subject. We can represent this case in two ways

1. Create table for both LECTURER and SUBJECT. Add the primary key of LECTURER in SUBJECT table as foreign key. It implies the lecturer name for that particular subject.
2. Create table for both LECTURER and SUBJECT. Add the primary key of SUBJECT in LECTURER table as foreign key. It implies the subject taught by the lecturer.

In both the case, meaning is same. Foreign key column can be added in either of the table, depending on the developer’s choice. 

* **Representing 1:N relationship**

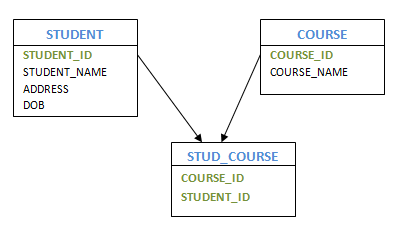
Consider SUBJECT and LECTURER relation, where each Lecturer teaches multiple subjects. This is a 1: N relation. In this case, primary key of LECTURER table is added to the SUBJECT table. i.e.; the primary key at 1 cardinality entity is added as foreign key to N cardinality entity



* **Representing M:N relationship**

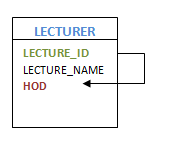
Consider the example, multiple students enrolled for multiple courses, which is M:N relation. In this case, we create STUDENT and COURSE tables for the entities. Create one more table for the relation ‘Enrolment’ and name it as STUD\_COURSE. Add the primary keys of COURSE and STUDENT into it, which forms the composite primary key of the new table.

That is, in this case both the participating entities are converted into tables, and a new table is created for the relation between them. Primary keys of entity tables are added into new table to form the composite primary key. We can add any additional columns, if present as attribute of the relation in ER diagram.



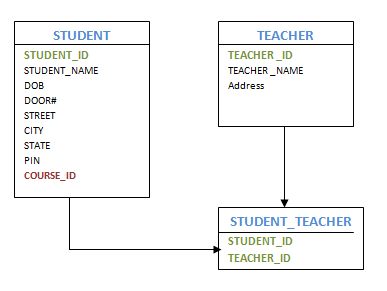
* **Self Referencing 1:N relation**

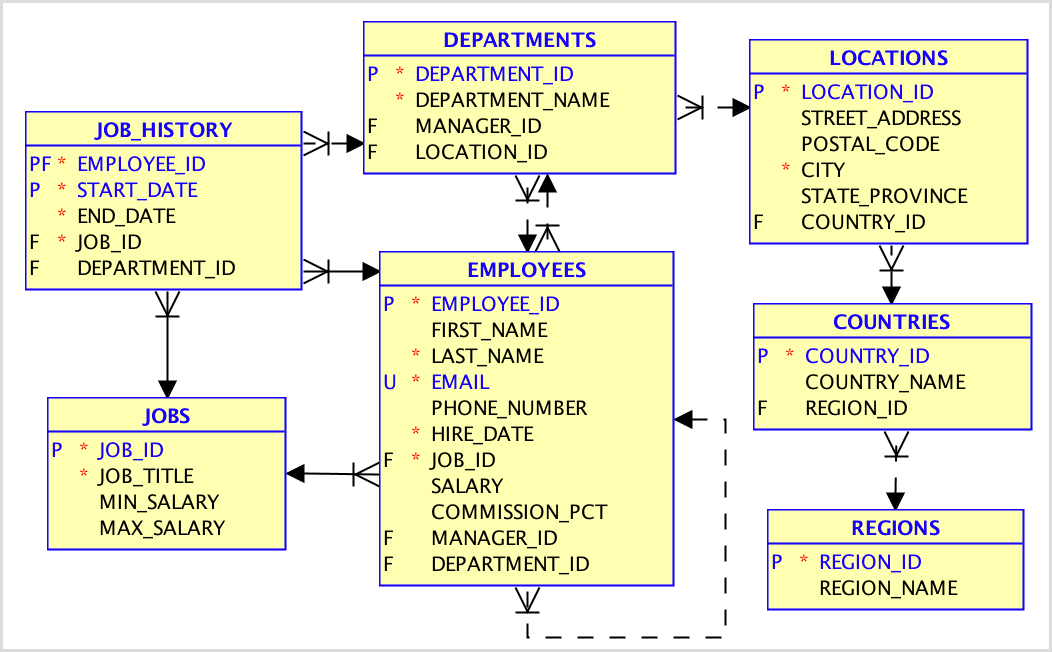
Consider the example of HOD and Lecturers. Here one of the Lecturers is a HOD of the department. i.e.; one HOD has multiple lecturers working with him. In this case, we create LECTURER table for the Lecturer entity. Create the columns and primary keys as usual. In order to represent HOD, we add one more column to LECTURER table which is same column as primary key, but acts as a foreign key. i.e.; LECTURER\_ID is the primary key of LECTURER table. We add one more column HOD, which will have LECTURER\_ID of the HOD. Hence LECTURER table will show HOD’s Lecturer ID for each Lecturer. In this case, primary key column acts as a foreign key in the same table.

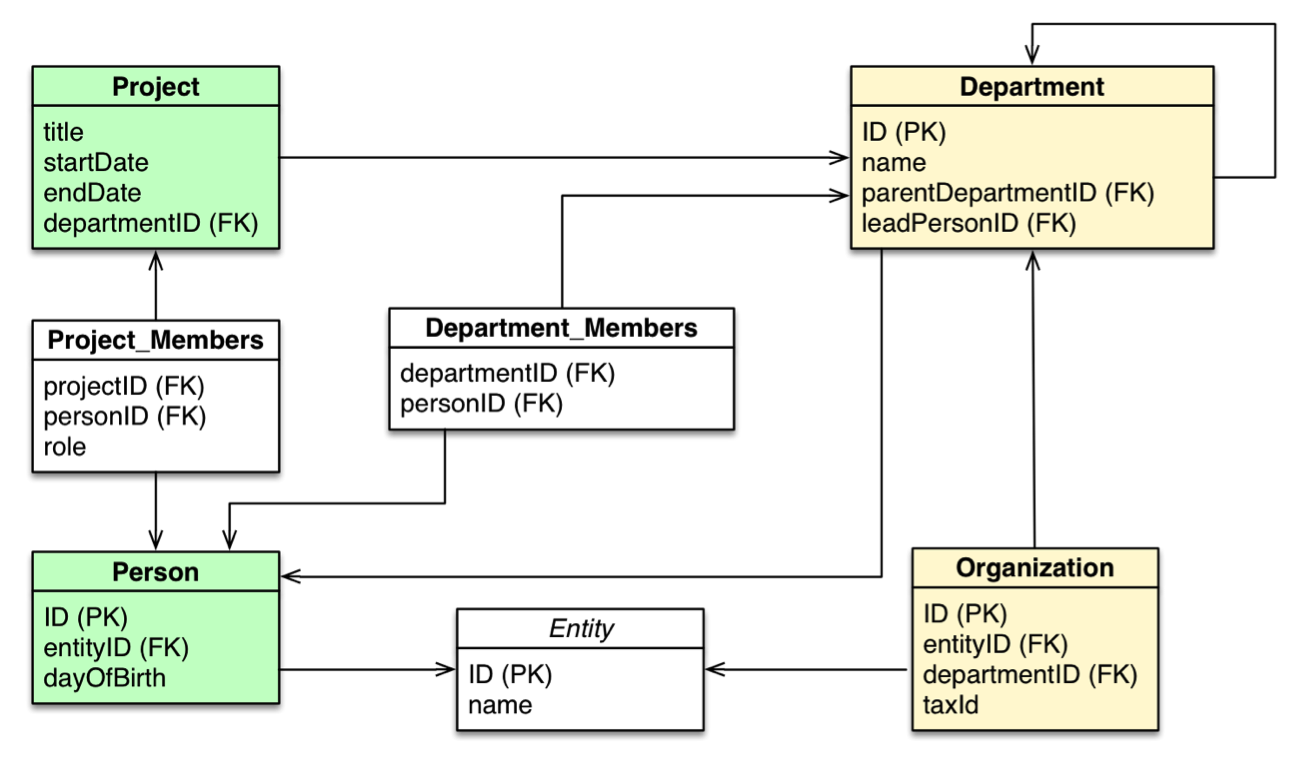


* **Self Referencing M:N relation**

Consider Student and Teacher example as ‘Many students have Many Teachers teaching the subjects’. Here relation between Student and Teacher is M:N. In this case, create independent tables for student and teacher, and set their primary keys. Then we create a new table for the relationship ‘have’ as STUDENT\_TEACHER, which will have student and teacher combination, and any other columns if applicable. Basically, student-teacher combination is the two primary key columns from respective tables, hence establishing the relationship between them. Both the primary keys from both tables act as a composite primary key in the new table. This reduces the storing of redundant data and consistency in the database.







**Query processing**: A 3-step process that transforms a high-level query (of relational calculus/SQL) into an **equivalent** and **more efficient** lower-level query (of relational algebra).

1. **Parsing and translation**

**–** Check syntax and verify relations.

**–** Translate the query into an equivalent relational algebra expression.

2. **Optimization**

**–** Generate an optimal evaluation plan (with lowest cost) for the query plan.

3. **Evaluation**

**–** The query-execution engine takes an (optimal) evaluation plan, executes that plan, and returns the answers to the query.



• The success of RDBMSs is due, in part, to the availability

**–** of declarative query languages that allow to easily express complex queries without knowing about the details of the physical data organization and

**–** of advanced query processing technology that transforms the high-level user/application queries into efficient lower-level query execution strategies.

• The query transformation should achieve both **correctness** and **efficiency –** The main difficulty is to achieve the efficiency

**–** This is also one of the most important tasks of any DBMS • **Distributed query processing**: Transform a high-level query (of relational calculus/SQL) on a distributed database (i.e., a set of global relations) into an **equivalent** and **efficient** lower-level query (of relational algebra) on relation fragments.

• Distributed query processing is more complex

**–** Fragmentation/replication of relations

**–** Additional communication costs

**–** Parallel execution

1. **RELATIONAL ALGEBRA:**

* The relational algebra is a procedural query language.
* Consists of set of operations that take one or two relation as input and return new relation as their result.
* Gives a procedural method of specifying a retrieval query.
* Forms the core component of a relational query engine.
* SQL queries are internally translated into Relational Algebra expressions.
* Provides a framework for query optimization.
* A sequence of relational algebra operations forms relational algebra expressions**.**
* The basic expressions of relational algebra consist of either relation in a database or a constant relation.
* A constant relation is written by listing its tuple within {}.
* A general expression in the relational algebra is constructed out of smaller sub expressions.

Let E1 and E2 be relational – algebra expressions, then relational algebra expressions are:

* E1 ᴜ E2
* E1- E2
* E1 × E2
* σp(E1), where P is a predicate on attribute in E1
* Πs(E1), where S is a list consisting of some of the attributes in E1
* ρx(E1), where X is the new name for the result of E1.

**8.1. FUNDAMENTAL OPERATIONS OF RELATIONAL ALGEBRA:**

* Select
* Project
* Union
* Set difference
* Cartesian product
* Rename

These are defined briefly as follows:

* + 1. **Select Operation (σ)**
* Selects tuples that satisfy the given predicate from a relation.
* Notation σ*p*(r)
* Where *p* stands for selection predicate and r stands for relation. *p* is prepositional logic formulae which may use connectors like and, or and not. These terms may use relational operators like: =, ≠, ≥, < ,  >,  ≤.

**For example:**

1. σ*subject="database"*(Books)

Output : Selects tuples from books where subject is 'database'.

1. σsubject="database" and price="450"(Books)

Output : Selects tuples from books where subject is 'database' and 'price' is 450.

1. σsubject="database" and price < "450" or year > "2010"(Books)

Output : Selects tuples from books where subject is 'database' and 'price' is 450 or the publication year is greater than 2010, that is published after 2010.

1. consider the following tables shown in Fig 1.8 where the first table gives the relation Person, the second table gives the result of \sigma_{Age \ge 34}( Person )and the third table gives the result of \sigma_{Age = Weight}( Person ).

|  |  |  |
| --- | --- | --- |
| Person | \sigma_{Age \ge 34}( Person ) | \sigma_{Age = Weight}( Person ) |
| |  |  |  | | --- | --- | --- | | Name | Age | Weight | | Harry | 34 | 80 | | Sally | 28 | 64 | | George | 29 | 70 | | Helena | 54 | 54 | | Peter | 34 | 80 | | |  |  |  | | --- | --- | --- | | Name | Age | Weight | | Harry | 34 | 80 | | Helena | 54 | 54 | | Peter | 34 | 80 | | |  |  |  | | --- | --- | --- | | Name | Age | Weight | | Helena | 54 | 54 | |

**Fig. 1.8.** Selection of tuples from Person relation

**8.1.2 PROJECT OPERATION (∏)**

* Projects column(s) that satisfy given predicate.
* Notation: ∏a1, a2, an (r)
* Where a1, a2 , an are attribute names of relation r.

Duplicate rows are automatically eliminated, as relation is a set.

**For example:**

1. ∏subject, author (Books)

Output : Selects and projects columns named as subject and author from relation Books.

1. Fig 1.9 shows the relation Person and its projection on the attributes Age and Weight.

|  |  |
| --- | --- |
| Person | \Pi_{Age,Weight}(Person) |
| |  |  |  | | --- | --- | --- | | Name | Age | Weight | | Harry | 34 | 180 | | Sally | 28 | 164 | | George | 29 | 170 | | Helena | 54 | 154 | | Peter | 34 | 180 | | |  |  | | --- | --- | | Age | Weight | | 34 | 180 | | 28 | 164 | | 29 | 170 | | 54 | 154 | |

**Fig.1.9.** Projection of the attributes age and weight on Person relation.

* + 1. **UNION OPERATION (∪)**
* Union operation performs binary union between two given relations and is defined as:
* Notion: r U s

r ∪ s = { t | t ∈ r or t ∈ s}

* Where r and s are either database relations or relation result set (temporary relation).
* For a union operation to be valid, the following conditions must hold:
* r, s must have same number of attributes.
* Attribute domains must be compatible.
* Duplicate tuples are automatically eliminated.

**For example:**

1. ∏ author (Books) ∪ ∏ author (Articles)

Output : Projects the name of author who has either written a book or an article or both.

1. STUDENT U INSTRUCTOR is illustrated in the figure 1.10.



**Fig. 1.10.** Example of union, intersection and set difference operations

* + 1. **SET DIFFERENCE ( − )**
* The result of set difference operation is tuples which are present in the first relation but are not in the second relation.
* Notation: r − s

r - s = { t | t ∈ r , but not in s}

* Where r and s are either database relations or relation result set (temporary relation).
* For a Set difference operation to be valid, the following conditions must hold:
* r, s must have same number of attributes.
* Attribute domains must be compatible.
* Find the tuples which are present in 'r' but are not in 's'.

**For example:**

1. ∏ author (Books) − ∏ author (Articles)

Output: Results the name of authors who has written books but not articles.

1. STUDENT - INSTRUCTOR and INSTRUCTOR - STUDENT are illustrated in the Figure 1.10.
   * 1. **CARTESIAN (OR CROSS) PRODUCT OPERATION**

* 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).
* Result is a relation Q 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.
* Hence, if R has nR tuples and S has nS tuples, then R x S will have nR \* nS tuples.
* The two operands do NOT have to be "type compatible‖.

**Example:**

**FEMALE\_EMPS ← σ SEX=’F’(EMPLOYEE)**

**EMPNAMES ← π FNAME, LNAME, SSN (FEMALE\_EMPS)**

**EMP\_DEPENDENTS ← EMPNAMES x DEPENDENT**

**EMP\_DEPENDENTS** will contain every combination of EMPNAMES and DEPENDENT.

This operation is illustrated in the figure 1.11.



**Fig 1.11.** Cartesian Product (Cross Product) operation

* + 1. **RENAME OPERATION ( ρ )**
* Results of relational algebra are also relations but with the different name.
* The rename operation allows us to rename the output relation.
* Denoted with small greek letter rho *ρ*
* Notation: *ρ* x (E), Where the result of expression E is saved with name of x.
* It is used to rename the attributes of a relation or the relation name or both.
* The general RENAME operation ρ can be expressed by any of the following forms:
  + - **ρS (B1, B2, …, Bn )(R) :** changes both the relation name to S, and the column (attribute) names to B1, B1, …..Bn
    - **ρS(R) :** changes the relation name only to S.
    - **ρ(B1, B2, …, Bn )(R) :** changesthe column (attribute) names only to B1, B1, …..Bn

**Example**

**TEMP ← σ DNO = 5 (EMPLOYEE)**

**ρ R(First\_name,Last\_name,Salary) [ π FNAME, LNAME, SALARY (TEMP) ]**

These two operations are illustrated in Figure 1.12



**Fig. 1. 12.** Results of Rename operation

* 1. **OTHER RELATIONAL ALGEBRA OPERATIONS**
* Set intersection
* Natural join
* Assignment

**8.2.1. SET INTERSECTION OPERATION:**

* INTERSECTION is denoted by ∩.
* The result of the operation R ∩ S, is a relation that includes all tuples that are in both R and 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.

**Example:**

STUDENT ∩ INSTRUCTOR is illustrated in the Figure 1.10.

**8.2.2. NATURAL JOIN ( ⋈ )**

* Natural join does not use any comparison operator.
* It does not concatenate the way Cartesian product does.
* Instead, Natural Join can only be performed if the there is at least one common attribute exists between relation.
* Those attributes must have same name and domain.
* Natural join acts on those matching attributes where the values of attributes in both relation is same.
* This operation is illustrated in Fig. 1.13.

|  |  |  |
| --- | --- | --- |
| Courses | | |
| CID | Course | Dept |
| CS01 | Database | CS |
| ME01 | Mechanics | ME |
| EE01 | Electronics | EE |

|  |  |
| --- | --- |
| HoD | |
| Dept | Head |
| CS | Alex |
| ME | Maya |
| EE | Mira |

|  |  |  |  |
| --- | --- | --- | --- |
| Courses ⋈ HoD | | | |
| Dept | CID | Course | Head |
| CS | CS01 | Database | Alex |
| ME | ME01 | Mechanics | Maya |
| EE | EE01 | Electronics | Mira |

**Fig.1.13.** Natural join of the Relation Courses, HoD (Courses ⋈ HoD)

**8.2.3. ASSIGNMENT OPERATION:**

* The assignment operation is more of a notational convenience rather than a real relational operation
* It is notated with the left-pointing arrow ←:
* variable ← E
* where E is any relational algebra expression.
* It helps in writing complex relational expressions using some procedures, so that they can be understood more easily.
* Write query as a sequential program consisting of a series of assignments followed by an expression whose value is displayed as a result of the query.
* Assignment must always be made to a temporary relation variable.

**Example:**

To compute r ÷ s

**temp1 ← ∏ R-S (r )**

**temp2 ← ∏ R-S ((temp1 x s) – ∏ R-S,S (r ) )**

**result = temp1 – temp2**

The result to the right of the ← is assigned to the relation variable on the left of the ←.