**AIMAN COLLEGE OF ARTS & SCIENCE FOR WOMEN,TIRUCHIRAPPALLI**

**Class : II B.Sc Computer Science – IV Semester**

**Subject Title : Database Systems**

**Subject Code : 16SCCCS4**

**Staff Name : A. Gurshith Begum**

**Designation : Asst. Professor**

**Introduction**

Database – Name itself indicates what it is. Database is a place/container where all the data is stored. But what is data?  In a database, even a smallest piece of information becomes data. For example, Student is a data, course is a data, and Color is a data, height, weight, food everything is data. In short, all the living and non-living objects in this world is a data.

Why we need data? We need data so that we can perform various actions on them. Say, we do not have any database and we want to record what is the height and weight of a baby in a year. What we do is note it in a piece of paper every month. At the end of the certain period or year, we would check up if he/she is growing correctly. If some entry is wrong or irrelevant, we correct it or strike it off. Same is done using database. We would be storing all these information in the database. If we want to check the growth, we would be pulling the information from the database, if we need to change any information, we can update/delete them. But all the data will be at one place – Database.

What type of data is stored in database? In a database, we would be grouping only related data together and storing them under one group name called table. This helps in identifying which data is stored where and under what name. It reduces the time to search for a particular data in a whole database.

And for whom these datas are stored? We store only related data – related to one particular requirement / application. For example, Student database – it will have all the information of students ranging from his ID, Name, Date of birth, class, to grade, prizes who are studying in a particular College.

How do we determine which data is relevant to be put in a particular database? It all depends on what database we are developing, and what is the exact requirement/purpose of it. Say, we need to create College database. What could college database contain? First thing is we need to store college information like its name, address. Next comes courses offered in that college, Staffs and their details, students and their details.  But do we store all these information under one table – College? Will database be quick in getting the data or updating? Certainly Not!  It would become a chaos if everything is stored in a single table. Hence they introduce certain rules to manage the database – relational database management system (RDBMS). RDBMS is a program that guides us how to create and maintain a database. It tells us how to divide related information into different tables and inter-relate them so that we can select/insert/update/delete all the related data easily and efficiently.

A database is a collection of different set of related data i.e.; mainly it is collection of tables. In the college example above, let us start identifying  groups of related data – College details, Courses,  Students, Staffs – lists goes on. Below diagram illustrates how do we divide the college data and put it in different tables.

If we observe COURSE table, all the information that related to course are listed under COURSE table. Similarly, STUDENT table has all the student information like his ID, Name -first, middle and last, DOB, age, his course, semester etc.  Each table has related information called attributes. In STUDENT table STUDENT\_ID is an attribute, STUDENT\_FIRST\_NAME is an attribute and so on. All these attribute together gives meaningful information about the student, called as Student record. i.e.; in a table each attribute acts as a column and their values become a row. Each row gives meaningful information about that table and it’s called as a record.

Now we have to decide what the related and relevant data for each table are. While designing a database, it is very important to design it correctly. Database design is like a foundation for a good database. A smallest mistake done at the design stage spoils the whole design, code and efficiency of the database.  A poor database design creates unwanted data in a table called data redundancy. Because of redundancy, unnecessarily database size increases and it also increases the chance of incorrect data while insertion, deletion and updation – leading to anamolies.  To understand it in a better way, consider the example of student table design.

A student has, say; his ID, Name, and Address, Subjects that he has opted for. The table structure for the same is designed as below.

By observing the STUDENT table design above, we identify that,

* Joseph has opted for two subjects – Mathematics and Physics. At the same time, his address is repeated for each of his subject, which is unnecessary data. It unnecessarily increases the table size and introduced a redundancy in the above table.
* Say, we have to update address of Joseph. In this case, we need to update all the entries of Joseph in the above table. Else his data will become inconsistent, leading to update anomaly.
* Imagine, student’s address is present in more than one table. IF we want to update any one of the student’s address, just updating Student table is not enough. We need to know which all table has address field which are related to Student. Not updating any one of the table will lead to incorrect data leading to update anamoly.
* Similarly, if any of the students drops out any of the subject and we need to delete his entry, entire entry of his would be deleted from above table. But he is still part of that college and whole of information is lost.   But this should not happen. We should have his information stored in database. This is called leading to delete anomaly.
* Also, if we need to enter any new student detail, which has not yet opted for any subject, then we need to enter all of his information with subject as NULL. Also, if a particular field, say address of a student is present in more than one table, we need to insert the data in all the tables. But for some of the tables, these data would be irrelevant at that point of time. All these situations lead to insertion anomaly.

Hence to avoid all these redundancies and anomalies, set of guidelines are introduced which is called Normalization.

**Normalization**

Normalization is a set of rules/guidelines/technique that is used while designing a database.  These rules help to remove all the anomalies and distribute the data among different related tables and query them efficiently and effectively. It removes all the duplication issues and incorrect data issues, helping to have a well designed database. Normalization is divided into following normal forms:

1. [**First Normal Form (1NF)**](https://www.tutorialcup.com/dbms/first-normal-form.htm)
2. [**Second  Normal Form (2NF)**](https://www.tutorialcup.com/dbms/second-normal-form.htm)
3. [**Third  Normal Form (3NF)**](https://www.tutorialcup.com/dbms/third-normal-form.htm)
4. [**Boyce-Codd Normal Form (3.5NF)**](https://www.tutorialcup.com/dbms/boyce-codd-normal-form.htm)
5. [**Forth Normal Form (4NF)**](https://www.tutorialcup.com/dbms/fourth-normal-form.htm)
6. [**Fifth Normal Form (5NF)**](https://www.tutorialcup.com/dbms/fifth-normal-form.htm)

**Note:**In real lifescenarios, we really do not apply all the normal forms to get the database design done. We usually apply till 3NF normal form.  4th and 5th normal forms are ignored most of the cases. It’s all depends on the necessity!!!

**First Normal Form (1NF)**

A table is said to be in First Normal Form (1NF) if and only if each attribute of the relation is atomic. That is,

* Each row in a table should be identified by primary key (a unique column value or group of unique column values)
* No rows of data should have repeating group of column values.

Let’s consider the STUDENT table with his ID, Name address and 2 subjects that he has opted for.



 Look at Chris entry. He has only subject. Hence Subject2 for him is NULL. Here storage space for second entry is simply wasted.

* In the case of Joseph, he has two subjects, Mathematics and Physics, both the columns have values. Imagine if he opts for third subject? There is no column for his third entry. In this case, whole table needs to be altered, which is not good at this stage. Once database is designed, it should be a perfect one. We should not be modifying it as we start adding/updating data.
* One of the requirements of 1NF is, each table should have primary key. This key in the table makes each record unique. In our example we have it already- STUDENT\_ID.
* Here, SUBJECT1 and SUBJECT2 are same set of columns, i.e.; it has same kind of information stored – Subject, which is a violation of first rule of 1NF. As it states, there should not be any repeating columns. We have to remove such columns. But think how?

In order to have STUDENT in 1NF, we have to remove multiple SUBJECT columns from STUDENT table. Instead, create only one SUBJECT column, and for each STUDENT enters as many rows as SUBJECT he has. After making this change, the above table will change as follows:



Now STUDENT\_ID alone cannot be a primary key, because it does not uniquely determines each record in the table. If we want to records for Joseph, and we query by his ID,100, gives us two records. Hence Student\_ID is no more a primary key.  When we observe the data in the table, all the four field uniquely determines each record. Hence all four fields together considered as primary key. Thus, the above table is in 1NF form.

**Second Normal Form (2NF)**

A relation is said to be in a second normal form if and only if,

* it’s in first normal form
* Every non-key attributes are  identified by the use of primary key
* All subset of data, which applies to have multiple rows in a table must be removed and placed in a new table. And this new table and the parent table should be related by the use of foreign key.

In the 1NF STUDENT table above, Joseph and Allen have multiple rows because of their SUBJECTS.  Although it is in 1NF form, it wastes storage space by repeating whole of their information – name and address in each row. In addition, Student ID alone is strong enough to be a primary key. If we make Student\_ID as primary, all other attributes in the table cannot be uniquely identified. This is because of multiple rows exists for single ID. Hence it does not satisfy second condition of 2NF.

So what we can do here is, apply the third condition of 2NF. Remove Subject from the STUDENT table and create a separate table for it. So the two tables are – STUDENT and SUBJECT. Now the STUDENT table will have only STUDENT information – STUDENT\_ID, STUDENT\_NAME and ADDRESS. New SUBJECT table will have STUDENT\_ID and SUBJECT\_NAME.



Now there is no repeating group of columns in STUDENT table and STUDENT\_ID is the primary key of STUDENT table. It uniquely identifies the Student name and address which are non key attributes of this table.  Hence it satisfies both 1NF and 2NF.

In the new SUBJECT table, Subject names opted by each student. Since same student cannot opt for same subject multiple times in academic year, there will not be any duplicity of data. But Student\_ID alone is not unique; hence it cannot be a primary key. Both Student\_ID and Subject\_Name is unique in this table. Hence both of them together become a primary key. Hence SUBJECT table satisfies 1NF.

There is no non-key attributes in SUBJECT table. Hence we cannot verify second condition of 2NF. According to the third condition of 2NF, we have removed the data which is forming multiple rows and put those details in new table. It also states that there should be relationship between the original table and new table by using foreign key constraint.  In the SUBJECT table, STUDENT\_ID is derived from STUDENT table. In this table, STUDENT \_ID is part of primary key as well as it is a foreign key. Hence we can easily relate both STUDENT and SUBJECT table by using STUDENT\_ID. Hence satisfies the third condition of 2NF.



If we want to know which all subjects Joseph has opted for, we would query as below:

SELECT std.STUDENT\_ID,

std.STUDENT\_NAME,

sb.SUBJECT

FROM STUDENT std, SUBJECT sb

WHERE std.STUDENT\_ID = sb.STUDENT\_ID

AND std.STUDENT\_NAME = 'Joseph';

**Third Normal Form (3NF)**

For a relation to be in third normal form:

* it should meet all the requirements of both 1NF and 2NF
* If there is any columns which are not related to primary key, then remove them and put it in a separate table, relate both the table by means of foreign key i.e.; there should not be any transitive dependency.

Let’s add three more columns – STREET, CITY and ZIP to STUDENT table to explain 3NF. Below is the table satisfies conditions for 2NF – there is primary key, no repeating columns, no duplicate datas.



Here, STREET and CITY have no relation with Student. It’s not directly related to student. They fully depend upon zip code. Since Student stays in that area, through zip code, street and city are related to him. This kind of relationship is called transitive dependency. Since its second level of dependency, it is not necessary to store these details in STUDENT table.

Similarly, if there are multiple students staying in same area, STUDENT table is having huge amount of records and there is a change requested for street or city name, then whole STUDENT table needs to be searched and updated. Imagine, we have to update ‘Fraser Village Drive’ to Fraser Village Dr’. The Update statement would be

UPDATE STUDENT std

SET std.STREET = 'Fraser Village Dr'

WHERE std .STREET = 'Fraser Village Drive';

Above query will search whole student table for ‘Fraser Village Drive’ and then update it to ‘Fraser Village Dr’. But searching a huge table and updating the single or multiple records will be a very time consuming, hence affecting the performance of the database.

Instead, if we have these details in a separate table ZIPCODE and is related to STUDENT table using zip? However ZIPCODE table will have comparatively less amount of records and we just have to update ZIPCODE table once. It will automatically reflect in the STUDENT table! Hence making the database and query simpler! And table is in 3NF.

UPDATE ZIPCODE z

SET z.STREET = 'Fraser Village Dr'

WHERE z .STREET = 'Fraser Village Drive';



Now if we have to select the whole address of a student, Chris, we join both STUDENT and ZIPCODE table using ZIP and get the whole address.

SELECT std.STUDENT\_NAME,

 z.STREET,

 z.CITY,

 z.ZIP

FROM STUDENT std, ZIPCODE z

WHERE std.ZIP = z.ZIP

AND std.STUDENT\_NAME = 'Chris';

**Boyce-Codd Normal Form (3.5NF)**

This normal form is also referred as 3.5 normal forms. This normal form

* Meets all the requirement of 3NF
* Any table is said to be in BCNF, if its candidate keys do not have any partial dependency on the other attributes. i.e.; in any table with (x, y, z) columns, if (x, y)->z and z->x then it’s a violation of BCNF. If (x, y) are composite keys and (x, y)->z, then there should not be any reverse dependency, directly or partially.

In the above 3NF example, STUDENT\_ID is the Primary key in STUDENT table and ZIP is the primary key in the ZIPCODE table. There is no other key column in each of the tables which determines the functional dependency. Hence it’s in BCNF form. That is, with STUDENT\_ID, we can retrieve STUDENT\_NAME and ZIP from STUDENT table. Similarly, with ZIP value, we can retrieve STREET and CITY from ZIPCODE table

Let us consider another example – consider each student who has taken major subjects has different advisory lecturers. Each student will have different advisory lecturers for same Subjects. There exists following relationship, which is violation of BCNF.

**(STUDENT\_ID, MAJOR\_SUBJECT) -> ADVISORY\_LECTURER**

**ADVISORY\_LECTURER -> MAJOR\_SUBJECT**

i.e. Major\_Subject which is a part of composite candidate key is determined non-key attribute of the same table, which is against the rule.

Below table will have all the anomalies too. If we delete any student from below table, it deletes lecturer’s information too. If we add any new lecturer/student to the database, it needs other related information also. Also, if we update subject for any student, his lecturer info also needs to be changed, else it will lead to inconsistency.



Hence we need to decompose the table so that eliminates so that it eliminates such relationship. Now in the new tables below, there are no inter-dependent composite keys (moreover, there is no composite key in both the tables). If we need to add/update/delete any lecturer, we can directly insert record into STUDENT\_ADVISOR table, without affecting STUDENT\_MAJOR table. If we need to insert/update/delete any subject for a student, then we can directly do it on STUDENT\_MAJOR table, without affecting STUDENT\_ADVISOR table.  When we have both advisor as well as major subject information, then we can directly add/update both the tables. Hence we have eliminated all the anamolies in the database.

**Fourth Normal Form (4NF)**

In the fourth normal form,

* It should meet all the requirement of 3NF
* Attribute of one or more rows in the table should not result in more than one rows of the same table leading to multi-valued dependencies

To understand it clearly, consider a table with Subject, Lecturer who teaches each subject and recommended Books for each subject.



If we observe the data in the table above it satisfies 3NF. But LECTURER and BOOKS are two independent entities here. There is no relationship between Lecturer and Books. In the above example, either Alex or Bosco can teach Mathematics. For Mathematics subject , student can refer either ‘Maths Book1’  or ‘Maths Book2’.  i.e.;

**SUBJECT –> LECTURER**

**SUBJECT–>BOOKS**

This is a multivalued dependency on SUBJECT. If we need to select both lecturer and books recommended for any of the subject, it will show up (lecturer, books) combination, which implies lecturer who recommends which book. This is not correct.

SELECT c.LECTURER, c.BOOKS FROM COURSE c WHERE SUBJECT = 'Mathematics';

To eliminate this dependency, we divide the table into two as below:



Now if we want to know the lecturer names and books recommended for any of the subject, we will fire two independent queries. Hence it removes the multi-valued dependency and confusion around the data. Thus the table is in 4NF.

--Select the lecturer names

SELECT c.SUBJECT , c.LECTURER FROM COURSE c WHERE c.SUBJECT = 'Mathematics';

--Select the recommended book names

SELECT c.SUBJECT , c.BOOKS FROM COURSE c WHERE c.SUBJECT = 'Mathematics';

## C:\Users\HP\Desktop\BOYCE2.pngFifth Normal Form (5NF)

A database is said to be in 5NF, if and only if,

* It’s in 4NF
* If we can decompose table further to eliminate redundancy and anomaly, and when we re-join the decomposed tables by means of candidate keys, we should not be losing the original data or any new record set should not arise. In simple words, joining two or more decomposed table should not lose records nor create new records.

Consider an example of different Subjects taught by different lecturers and the lecturers taking classes for different semesters.

**Note**: Please consider that Semester 1 has Mathematics, Physics and Chemistry and Semester 2 has only Mathematics in its academic year!!



In above table, Rose takes both Mathematics and Physics class for Semester 1, but she does not take Physics class for Semester 2.  In this case, combination of all these 3 fields is required to identify a valid data. Imagine we want to add a new class – Semester3 but do not know which Subject and who will be taking that subject. We would be simply inserting a new entry with Class as Semester3 and leaving Lecturer and subject as NULL. As we discussed above, it’s not a good to have such entries. Moreover, all the three columns together act as a primary key, we cannot leave other two columns blank!

Hence we have to decompose the table in such a way that it satisfies all the rules till 4NF and when join them by using keys, it should yield correct record. Here, we can represent each lecturer’s Subject area and their classes in a better way. We can divide above table into three – (SUBJECT, LECTURER), (LECTURER, CLASS), (SUBJECT, CLASS)



Now, each of combinations is in three different tables. If we need to identify who is teaching which subject to which semester, we need join the keys of each table and get the result.

For example, who teaches Physics to Semester 1, we would be selecting Physics and Semester1 from table 3 above, join with table1 using Subject to filter out the lecturer names. Then join with table2 using Lecturer to get correct lecturer name. That is we joined key columns of each table to get the correct data. Hence there is no lose or new data – satisfying 5NF condition.

SELECT t3.Class, t3.Subject, t1.Lecturer

FROM TABLE3 t3, TABLE3 t2, TABLE3 t1,

where t3.Class = 'SEMESTER1' and t3.SUBJECT= 'PHYSICS'

AND t3.Subject = t1.Subject

AND t3.Class = t2.Class

 AND t1.Lecturer = t2.Lecturer;

Relational Calculus

Relational calculus is a non procedural query language. It uses mathematical predicate calculus instead of algebra. It provides the description about the query to get the result where as relational algebra gives the method to get the result. It informs the system what to do with the relation, but does not inform how to perform it.

For example, steps involved in listing all the students who attend ‘Database’ Course in relational algebra would be

* SELECT the tuples from COURSE relation with COURSE\_NAME = ‘DATABASE’
* PROJECT the COURSE\_ID from above result
* SELECT the tuples from STUDENT relation with COUSE\_ID resulted above.

In the case of relational calculus, it is described as below:

Get all the details of the students such that each student have course as ‘Database’.

See the difference between relational algebra and relational calculus here. From the first one, we are clear on how to query and which relations to be queried. But the second tells what needs to be done to get the students with ‘database’ course. But it does tell us how we need to proceed to achieve this. Relational calculus is just the explanative way of telling the query.

There are two types of relational calculus – Tuple Relational Calculus (TRC) and Domain Relational Calculus (DRC).

Tuple Relational Calculus

A tuple relational calculus is a non procedural query language which specifies to select the tuples in a relation. It can select the tuples with range of values or tuples for certain attribute values etc. The resulting relation can have one or more tuples. It is denoted as below:

**{t | P (t)}**or **{t | condition (t)} —** this is also known as expression of relational calculus

Where t is the resulting tuples, P(t) is the condition used to fetch t.

**{t | EMPLOYEE (t) and t.SALARY>10000}**  – implies that it selects the tuples from EMPLOYEE relation such that resulting employee tuples will have salary greater than 10000. It is example of selecting a range of values.

**{t | EMPLOYEE (t) AND t.DEPT\_ID = 10}** – this select all the tuples of employee name who work for Department 10.

The variable which is used in the condition is called **tuple variable**. In above example t.SALARY and t.DEPT\_ID are tuple variables. In the first example above, we have specified the condition t.SALARY >10000. What is the meaning of it? For all the SALARY>10000, display the employees. Here the SALARY is called as bound variable. Any tuple variable with ‘*For All’* (?) or ‘*there exists’* (?) condition is **called bound variable**. Here, for any range of values of SALARY greater than 10000, the meaning of the condition remains the same. Bound variables are those ranges of tuple variables whose meaning will not change if the tuple variable is replaced by another tuple variable.

In the second example, we have used DEPT\_ID= 10. That means only for DEPT\_ID = 10 display employee details. Such variable is called free variable. Any tuple variable without any ‘*For All’* or ‘*there exists’* condition is called **Free Variable**. If we change DEPT\_ID in this condition to some other variable, say EMP\_ID, the meaning of the query changes. For example, if we change EMP\_ID = 10, then above it will result in different result set. Free variables are those ranges of tuple variables whose meaning will change if the tuple variable is replaced by another tuple variable.

All the conditions used in the tuple expression are called as well formed formula – WFF. All the conditions in the expression are combined by using logical operators like AND, OR and NOT, and qualifiers like ‘For All’ (?) or ‘there exists’ (?). If the tuple variables are all bound variables in a WFF is called **closed WFF**. In an **open WFF**, we will have at least one free variable.

Domain Relational Calculus

In contrast to tuple relational calculus, domain relational calculus uses list of attribute to be selected from the relation based on the condition. It is same as TRC, but differs by selecting the attributes rather than selecting whole tuples. It is denoted as below:

{< a1, a2, a3, … an > | P(a1, a2, a3, … an)}

Where a1, a2, a3, … an are attributes of the relation and P is the condition.

For example, select EMP\_ID and EMP\_NAME of employees who work for department 10

**{<EMP\_ID, EMP\_NAME> | <EMP\_ID,** **EMP\_NAME> ? EMPLOYEE Λ DEPT\_ID = 10}**

Get name of the department name that Alex works for.

**{DEPT\_NAME |< DEPT\_NAME > ? DEPT Λ ? DEPT\_ID (<DEPT\_ID> ? EMPLOYEE Λ EMP\_NAME = Alex)}**

Here green color expression is evaluated to get the department Id of Alex and then it is used to get the department name form DEPT relation.

Let us consider another example where select EMP\_ID, EMP\_NAME and ADDRESS the employees from the department where Alex works. What will be done here?

**{<EMP\_ID, EMP\_NAME, ADDRESS, DEPT\_ID > | <EMP\_ID, EMP\_NAME, ADDRESS, DEPT\_ID> ? EMPLOYEE Λ ? DEPT\_ID (<DEPT\_ID> ? EMPLOYEE Λ EMP\_NAME = Alex)}**

First, formula is evaluated to get the department ID of Alex (green color), and then all the employees with that department is searched (red color).

Other concepts of TRC like free variable, bound variable, WFF etc remains same in DRC too. Its only difference is DRC is based on attributes of relation.

Introduction

ER Data Model is based on the real world objects and their relationship. In other words, each and everything, either living or non-living things in this world forms the object in database world. We identify all the required objects for our database requirement and give the shape of database objects. Before putting them into database, it is very much essential to understand the requirement properly and design them efficiently. It is like a foundation of the building. For this purpose, we use ER diagrams where we plan the database pictorially. ER diagram basically breaks requirement into entities, attributes and relationship. Let us see them in detail.

Entity

In a database, we would be grouping only related data together and storing them under one group name called Entity / Table. This helps in identifying which data is stored where and under what name. It reduces the time to search for a particular data in a whole database. Say, we are creating a School database. Then what all things come into our mind? It is school, students, teachers, subjects, class etc. If you observe what we have listed are items/names which are part of school, but each of them having their own group of related information? i.e.; Student has ID, name, address, DOB, class in which he is studying etc. for a Student. Similarly, TEACHERS will have their ID, name, address subjects that they are teaching, class which they are teaching etc. Hence each entity forms a group of related information. In real world each and every object forms an entity.

In short, all the living and non-living things in the world form entity. One can consider all nouns as entities, if that makes it simpler.

* **Strong Entity**: Entities having its own attribute as primary keys are called strong entity. For example, STUDENT has STUDENT\_ID as primary key. Hence it is a strong entity.
* **Weak Entity**: Entities which cannot form their own attribute as primary key are known weak entities. These entities will derive their primary keys from the combination of its attribute and primary key from its mapping entity.

Consider CLASS and SECTION entity. The SECTION has SECTION \_ID and NAME as its attribute. But SECTION \_ID alone cannot be a primary key, since it fails to tell for which course it is related to. We will not be uniquely identifying the course section by this attribute alone. But if this attribute along with CLASS\_ID gives the meaning for each section and we can uniquely identify the sections.



* **Composite Entity**: Entities participating in the many to many relationships are called composite entity. In this case, apart from two entities that are part of relation, we will one more hidden entity in the relation. We will be creating a new entity with the relation, and create a primary key by using the primary keys of other two entities. Consider the example, multiple students enrolled for multiple courses. In this case, we create STUDENT and COURSE. Then we 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.



* **Recursive Entity**: If a relation exists between the same entities, then such entities are called as recursive entity. For example, mapping between manager and employee is recursive entity. Here manager is mapped to the same entity Employee. HOD of the department is another example of having recursive entity.



Attribute

In the above example of STUDENT entity, Student has ID, name, address, DOB, class in which he is studying are all called as attribute of STUDENT entity. It’s also known as columns of the table. In other words, an attribute is a list of all related information of an entity, which has valid value.

An attribute can have single value or multiple value or range of values. In addition, each attribute can contain certain type of data like only numeric value, or only alphabets, or combination of both, or date or negative or positive values etc. Depending on the values that an attribute can take, it is divided into different types.

* **Simple Attribute**These kinds of attributes have values which cannot be divided further. For example, STUDENT\_ID attribute which cannot be divided further. Passport Number is unique value and it cannot be divided.
* **Composite Attribute**This kind of attribute can be divided further to more than one simple attribute. For example, address of a person. Here address can be further divided as Door#, street, city, state and pin which are simple attributes.
* **Derived Attribute**Derived attributes are the one whose value can be obtained from other attributes of entities in the database. For example, Age of a person can be obtained from date of birth and current date. Average salary, annual salary, total marks of a student etc are few examples of derived attribute.
* **Stored Attribute**The attribute which gives the value to get the derived attribute are called Stored Attribute. In example above, age is derived using Date of Birth. Hence Date of Birth is a stored attribute.
* **Single Valued Attribute**These attributes will have only one value. For example, EMPLOYEE\_ID, passport#, driving license#, SSN etc have only single value for a person.
* **Multi-Valued Attribute**These attribute can have more than one value at any point of time. Manager can have more than one employee working for him, a person can have more than one email address, and more than one house etc is the examples.
* **Simple Single Valued Attribute**This is the combination of above four types of attributes. An attribute can have single value at any point of time, which cannot be divided further. For example, EMPLOYEE\_ID – it is single value as well as it cannot be divided further.
* **Simple Multi-Valued Attribute**Phone number of a person, which is simple as well as he can have multiple phone numbers is an example of this attribute.
* **Composite Single Valued Attribute**Date of Birth can be a composite single valued attribute. Any person can have only one DOB and it can be further divided into date, month and year attributes.
* **Composite Multi-Valued Attribute**Shop address which is located two different locations can be considered as example of this attribute.
* **Descriptive Attribute**Attributes of the relationship is called descriptive attribute. For example, employee works for department. Here ‘works for’ is the relation between employee and department entities. The relation ‘works for’ can have attribute DATE\_OF\_JOIN which is a descriptive attribute.

Keys

Keys are the attributes of the entity, which uniquely identifies the record of the entity. For example STUDENT\_ID identifies individual students, passport#, license # etc.

As we have seen already, there are different types of keys in the database.

**Super Key** is the one or more attributes of the entity, which uniquely identifies the record in the database.

**Candidate Key** is one or more set of keys of the entity. For a person entity, his SSN, passport#, license# etc can be a super key.

**Primary Key** is the candidate key, which will be used to uniquely identify a record by the query. Though a person can be identified using his SSN, passport# or license#, one can choose any one of them as primary key to uniquely identify a person. Rest of them will act as a candidate key.



**Foreign Key** of the entity attribute in the entity which is the primary key of the related entity. Foreign key helps to establish the mapping between two or more entities.



Relationship

A relationship defines how two or more entities are inter-related. For example, STUDENT and CLASS entities are related as ‘Student X **studies** in a Class Y’. Here ‘Studies’ defines the relationship between Student and Class. Similarly, Teacher and Subject are related as ‘Teacher A **teaches** Subject B’. Here ‘teaches’ forms the relationship between both Teacher and Subject.

Degrees of Relationship

In a relationship two or more number of entities can participate. The number of entities who are part of a particular relationship is called degrees of relationship. If only two entities participate in the mapping, then degree of relation is 2 or binary. If three entities are involved, then degree of relation is 3 or ternary. If more than 3 entities are involved then the degree of relation is called n-degree or n-nary.

Cardinality of Relationship

How many number of instances of one entity is mapped to how many number of instances of another entity is known as cardinality of a relationship. In a ‘studies’ relationship above, what we observe is only one Student X is studying in on Class Y. i.e.; single instance of entity student mapped to a single instance of entity Class. This means the cardinality between Student and Class is 1:1.

**Based on the cardinality, there are 3 types of relationship.**

* **One-to-One (1:1)**: As we saw in above example, one instance of the entity is mapped to only one instance of another entity.Consider, HOD of the Department. There is only one HOD in one department. That is there is 1:1 relationship between the entity HOD and Department.



* **One-to-Many (1: M)**: As we can guess now, one to many relationship has one instance of entity related to multiple instances of another entity. One manager manages multiple employees in his department. Here Manager and Employee are entities, and the relationship is one to many. Similarly, one teacher teaches multiple classes is also a 1: M relationship.

**Many-to-Many (M: N)**: This is a relationship where multiple instances of entities are related to multiple instances of another entity. A relationship between TEACHER and STUDENT is many to many. How? Multiple Teachers teach multiple numbers of Students. Similarly, above example of 1:1 can be M:N !! Surprised?? Yes, it can be M:N relationship, provided, how we relate these two entities. Multiple Students enroll for multiple classes/courses makes this relationship M:N. The relationship ‘studies’ and ‘enroll’ made the difference here. That means, it all depends on the requirement and how we are relating the entities.



**Participation Constraints**

This represents how an entity is involved in the relation. That means, if all the entity values are participating in any relation, then it is called total participation. If only few values of an entity are part of relation, then it is a partial participation.

For example, ‘Employee Works for a Department’. Here all the employees work for one or the other department. No employees are left without department. Hence all the employees are participating in this relation. Thus, participation of employee is total participation.

But individual Department will not have all the employees. Each department will have only few groups of employees working. Hence partial participation of department is seen here

**Introduction**

We have seen what ER diagram is and what its basic concepts are. Now let us see how to draw the ER diagram using these concepts. This diagram is the first step in designing the database.

ER Diagram Symbols

Since ER diagram is the pictorial representation of real world objects, it involves various symbols and notation to draw the diagrams. Let us see one by one below.

**Entity**: Rectangles are used to represent the entity in the diagram. Name of the Entity is written inside the rectangle.



A strong entity is represented by simple rectangle as shown above. A weak entity is represented by two rectangles as shown below.



**Attribute**: An oval shape is used to represent the attributes. Name of the attribute is written inside the oval shape and is connected to its entity by a line.

Multivalued attributes are represented by double oval shape; where as derived attributes are represented by oval shape with dashed lines. A composite attribute is also represented by oval shape, but these attribute will be connected to its parent attribute forming a tree structure.



**Primary Key**: An underline to the attribute name is put to represent the primary key. The key attribute of the weak entity is represented by dashed underline.



**Relationship**: A diamond shape is used to show the relationship between the entities. A mapping with weak entity is shown using double diamond. Relationship name will be written inside them.



**Cardinality of Relationship**: Different developers use different notation to represent the cardinality of the relationship. Not only for cardinality, but for other objects in ER diagram will have slightly different notations. But main difference is noticed in the cardinality. For not to get confused with many, let us see two types of notations for each.

**One-to-one relation: –** A one-to-one relationship is represented by adding ‘1’ near the entities on the line joining the relation. In another type of notation one dash is added to the relationship line at both ends.



**One-to-Many relation**: A one-to-many relationship is represented by adding ‘1’ near the entity at left hand side of relation and ‘N’ is written near the entity at right side. Other type of notation will have dash at LHS of relation and three arrow kind of lines at the RHS of relation as shown below.



**Many-to-Many relation**: A one-to-many relationship is represented by adding ‘M’ near the entity at left hand side of relation and ‘N’ is written near the entity at right side. Other type of notation will have three arrow kinds of lines at both sides of relation as shown below.



**Participation Constraints**: Total participation constraints are shown by double lines and partial participations are shown as single line.



Complete ER diagram

Let us create a simple ER diagram for a STUDENT database. What is the requirement of this database?

‘Student attends class. Each class is divided into one or more sections. Each class will have its own specified subjects. Students have to attend all the subjects of the class that he attends’.

Now let us identify what are the entities? STUDENT, CLASS, SECTION, SUBJECT are the entities. Attributes of these entities are not specified here. But we know what could be the entities of each of the entities. We can list them as below at this point of time.

|  |  |  |  |
| --- | --- | --- | --- |
| STUDENT | CLASS | SECTION | SUBJECT |
| STUDENT\_ID | CLASS\_ID | SECTION\_ID | SUBJECT\_ID |
| STUDENT\_NAME | CLASS\_NAME | SECTION\_NAME | SUBJECT\_NAME |
| ADDRESS |  |  |  |
| DOB |  |  |  |
| AGE |  |  |  |
| CLASS\_ID |  |  |  |
| SECTION\_ID |  |  |  |

What are the relationships we have? ‘Attends’, ‘has section’, ‘have subjects’ and ‘studies subjects’ are the relations here. With this knowledge of requirement, we can draw the ER diagram as below.



Observe the diagram carefully. Did we miss or drew it correctly? Are we missing anything on the diagram? Is it inferring correct requirement? What are our observations?

* Age attribute can be derived from DOB. Hence we have to draw dashed oval.
* Address is a composite attribute. We have to draw its sub attributes too. So that we will be very clear about his address details.
* If we see the SECTION entity, by section id, will we be able get the section that student attends? There is no relation mentioned between Student and Section. But Section is mapped only with Class. What do we understand from this? Section is a weak entity. Hence we have to represent it properly.
* If we look at ‘attends’ relationship between STUDENT and CLASS, we can have ‘Joining Date’ and ‘Total Number of Hours’ attributes. But it is an attribute of relation. We have to show them in the diagram.
* Since each class is having different subjects and Students attends those subjects, we can modify the relation ‘studies’ to ‘has’ relation on the **relation** ‘attends’.

Now the diagram will change to reflect all above points.



Are done with complete diagram? We have not added the cardinality and participation in the diagram.

**What are the participation constraints here?**

* All the Students attend any one of the class, but class can have only certain group of students. Hence total participation of Students and partial participation of class in ‘Attends’ relation.
* All the class has section and all the section has class. Hence both are total participation.
* All the Students study some of the subjects specific for their class and each class has only some group of subjects. Hence partial participation of both STUDENT and CLASS. Each subject will be studied by some students and it will be part of some class. Hence this also partial participation.

**What are the cardinalities of all the relationship?**

* Each Student attends only one class at a time. Hence it is a **1: 1** relation.
* Each class has one or more sections. Hence it can be considered as **1: N** relation.
* Each student attends many subjects and each class has many subjects. Hence it is a **1:N** relation.

**Note:** If you look at STUDENT and CLASS relationship as many Students attend one class, then it would be an **M: 1** relation. It is all up to the developer, how he looks at the requirement.



Now it is a complete ER diagram for simple Student database.