

EXPERIMENT – 1

Entity Relationship Model

Entity-Relationship model is used to represent a logical design of a database to be created. In ER model, real world objects (or concepts) are abstracted as entities, and different possible associations among them are modeled as relationships.

For example, student and school -- they are two entities. Students study in school. So, these two entities are associated with a relationship "Studies in". As another example, consider a system where some job runs every night, which updates the database. Here, job and database could be two entities. They are associated with the relationship "Updates".

Entity Set and Relationship Set

An entity set is a collection of all similar entities. For example, "Student" is an entity set that abstracts all students. Ram, John are specific entities belonging to this set. Similarly, a "Relationship" set is a set of similar relationships.

Attributes of Entity

Attributes are the characteristics describing any entity belonging to an entity set. Any entity in a set can be described by zero or more attributes.

For example, any student has got a name, age, an address. At any given time, a student can study only at one school. In the school he would have a roll number, and of course a grade in which he studies. These data are the attributes of the entity set Student.

Mapping Cardinalities

One of the main tasks of ER modeling is to associate different entity sets. Let us consider two entity sets E1 and E2 associated by a relationship set R. Based on the number of entities in E1 and E2 are associated with, we can have the following four type of mappings:



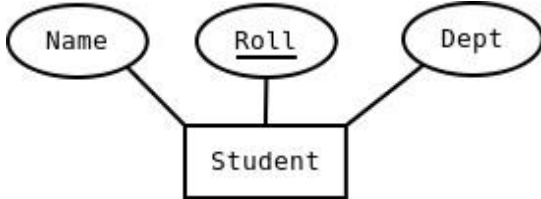

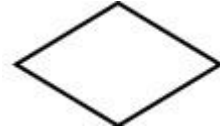

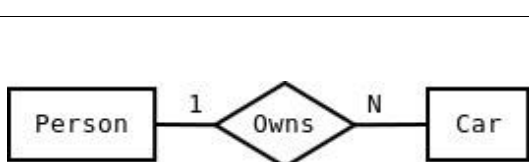

- One to one: An entity in E1 is related to at most a single entity in E2, and vice versa
- One to many: An entity in E1 could be related to zero or more entities in E2. Any entity in E2 could be related to at most a single entity in E1.
- Many to one: Zero or more number of entities in E1 could be associated to a single entity in E2. However, an entity in E2 could be related to at most one entity in E1.
- Many to many: Any number of entities could be related to any number of entities in E2, including zero, and vice versa.

ER Diagram

From a given problem statement we identify the possible entity sets, their attributes, and relationships among different entity sets. Once we have this information, we represent them pictorially, called an entity-relationship (ER) diagram



Graphical Notations for ER Diagram

Term	Notation	Remarks
Entity set		Name of the set is written inside the rectangle
Attribute		Name of the attribute is written inside the ellipse
Entity with attributes		Roll is the primary key; denoted with an underline
Weak entity set		
Relationship set		Name of the relationship is written inside the diamond
Related entity sets		
Relationship cardinality		A person can own zero or more cars but no two persons can own the same car
Relationship with weak entity set		

Importance of ER modeling

Figure - 01 shows the different steps involved in implementation of a (relational) database.

Figure - 01: Steps to implement a RDBMS

Given a problem statement, the first step is to identify the entities, attributes, and relationships. We represent them using an ER diagram. Using this ER diagram, table structures are created, along with required constraints. Finally, these tables are normalized to remove redundancy and maintain data integrity. Thus, to have data stored efficiently, the ER diagram is to be drawn as much detailed and accurate as possible.

Description of Normalization

Normalization is the process of organizing data in a database. This includes creating tables and establishing relationships between those tables according to rules designed both to protect the data and to make the database more flexible by eliminating redundancy and inconsistent dependency.

Redundant data wastes disk space and creates maintenance problems. If data that exists in more than one place must be changed, the data must be changed in exactly the same way in all locations. A customer address change is much easier to implement if that data is stored only in the Customers table and nowhere else in the database.

What is an "inconsistent dependency"? While it is intuitive for a user to look in the Customers table for the address of a particular customer, it may not make sense to look there for the salary of the employee who calls on that customer. The

employee's salary is related to, or dependent on, the employee and thus should be moved to the Employees table. Inconsistent dependencies can make data difficult to access because the path to find the data may be missing or broken.

There are a few rules for database normalization. Each rule is called a "normal form." If the first rule is observed, the database is said to be in "first normal form." If the first three rules are observed, the database is considered to be in "third normal form." Although other levels of normalization are possible, third normal form is considered the highest level necessary for most applications.

As with many formal rules and specifications, real world scenarios do not always allow for perfect compliance. In general, normalization requires additional tables and some customers find this cumbersome. If you decide to violate one of the first three rules of normalization, make sure that your application anticipates any problems that could occur, such as redundant data and inconsistent dependencies.

The following descriptions include examples.

First Normal Form

- Eliminate repeating groups in individual tables.
- Create a separate table for each set of related data.
- Identify each set of related data with a primary key.

Do not use multiple fields in a single table to store similar data. For example, to track an inventory item that may come from two possible sources, an inventory record may contain fields for Vendor Code 1 and Vendor Code 2.

What happens when you add a third vendor? Adding a field is not the answer; it requires program and table modifications and does not smoothly accommodate a dynamic number of vendors. Instead, place all vendor information in a separate table called Vendors, then link inventory to vendors with an item number key, or vendors to inventory with a vendor code key.

Second Normal Form

- Create separate tables for sets of values that apply to multiple records.
- Relate these tables with a foreign key.

Records should not depend on anything other than a table's primary key (a compound key, if necessary). For example, consider a customer's address in an accounting system. The address is needed by the Customers table, but also by the Orders, Shipping, Invoices, Accounts Receivable, and Collections tables. Instead of storing the customer's address as a separate entry in each of these tables, store it in one place, either in the Customers table or in a separate Addresses table.

Third Normal Form

- **Eliminate fields that do not depend on the key.**

Values in a record that are not part of that record's key do not belong in the table. In general, any time the contents of a group of fields may apply to more than a single record in the table, consider placing those fields in a separate table.

For example, in an Employee Recruitment table, a candidate's university name and address may be included. But you need a complete list of universities for group mailings. If university information is stored in the Candidates table, there is no way to list universities with no current candidates. Create a separate Universities table and link it to the Candidates table with a university code key.

EXCEPTION: Adhering to the third normal form, while theoretically desirable, is not always practical. If you have a Customers table and you want to eliminate all possible interfiled dependencies, you must create separate tables for cities, ZIP codes, sales representatives, customer classes, and any other factor that may be duplicated in multiple records. In theory, normalization is worth pursuing. However, many small tables may degrade performance or exceed open file and memory capacities.

It may be more feasible to apply third normal form only to data that changes frequently. If some dependent fields remain, design your application to require the user to verify all related fields when any one is changed.

Other Normalization Forms

Fourth normal form, also called Boyce Codd Normal Form (BCNF), and fifth normal form do exist, but are rarely considered in practical design. Disregarding these rules may result in less than perfect database design, but should not affect functionality.

Normalizing an Example Table

These steps demonstrate the process of normalizing a fictitious student table.

1. Unnormalized table:

Student#	Advisor	Adv-Room	Class1	Class2	Class3
1022	Jones	412	101-07	143-01	159-02
4123	Smith	216	201-01	211-02	214-01

2. First Normal Form: No Repeating Groups

Tables should have only two dimensions. Since one student has several classes, these classes should be listed in a separate table. Fields Class1, Class2, and Class3 in the above records are indications of design trouble.

Spreadsheets often use the third dimension, but tables should not. Another way to look at this problem is with a one-to-many relationship, do not put the one side and the many sides in the same table. Instead, create another table in first normal form by eliminating the repeating group (Class#), as shown below:

Student#	Advisor	Adv-Room	Class#
1022	Jones	412	101-07
1022	Jones	412	143-01
1022	Jones	412	159-02
4123	Smith	216	201-01
4123	Smith	216	211-02
4123	Smith	216	214-01

3. Second Normal Form: Eliminate Redundant Data

Note the multiple Class# values for each Student# value in the above table. Class# is not functionally dependent on Student# (primary key), so this relationship is not in second normal form.

The following two tables demonstrate second normal form:

Students:

Student#	Advisor	Adv-Room
1022	Jones	412
4123	Smith	216

Registration:

Student#	Class#
1022	101-07
1022	143-01
1022	159-02
4123	201-01
4123	211-02
4123	214-01

4. Third Normal Form: Eliminate Data Not Dependent on Key

In the last example, Adv-Room (the advisor's office number) is functionally dependent on the Advisor attribute. The solution is to move that attribute from the Students table to the Faculty table, as shown below:

Students:

Student	Advisor
1022	Smith
4023	Jones

Faculty:

Name	Room	Dept
Jones	412	42
Smith	216	42

SQL is structure query language. SQL contains different data types those are

1. char(size)
2. varchar2(size)
3. date
4. number (p, s)
5. long
6. raw/long raw

How to Write and execute sql, pl/sql commands/programs:

1). Open your oracle application by the following navigation

Start->all programs->oracle orahome.->application

development->sql.

2). You will be asked for user name, password and host string

You have to enter user name, pass word and host string as given by the administrator. It will be different from one user to another user.

3). Upon successful login you will get SQL prompt (SQL>).

In two ways you can write your programs:

- a. directly at SQL prompt
- b. or in sql editor.

If you type your programs at sql prompt then screen will look like follow: SQL> SELECT ename,empno,sal from emp;

where 2 and 3 are the line numbers and rest is the command /program.....

to execute above program/command you have to press '/' then enter.

Here editing the program is somewhat difficult; if you want to edit the previous command then you have to open sql editor (by default it displays the sql buffer contents). By giving 'ed' at sql prompt.(this is what I mentioned as a second method to type/enter the program). in the sql editor you can do all the formatting/editing/file operations directly by selecting menu options provided by it.

To execute the program which saved; do the following

SQL> @ programname.sql

Or

SQL> Run programname.sql

Then press '\` key and enter.

This how we can write, edit and execute the sql command and programs. Always you have to save your programs in your own logins.