MODULE II

Introduction to the Relational Model

Relational Model was proposed by E.F. Codd to model data in the form of relations or tables. After designing the conceptual model of Database using ER diagram, we need to convert the conceptual model in the relational model which can be implemented using any RDBMS languages like Oracle SQL, MySQL etc. So we will see what Relational Model is.

What is Relational Model?

Relational Model represents how data is stored in Relational Databases. A relational database stores data in the form of relations (tables). Consider a relation STUDENT with attributes ROLL_NO, NAME, ADDRESS, PHONE and AGE shown in Table 1.

STUDENT

ROLL_NO	**NAME**	**ADDRESS**	**PHONE**	**AGE**
1	RAM	DELHI	9455123451	18
2	RAMESH	GURGAON	9652431543	18
3	SUJIT	ROHTAK	9156253131	20
4	SURESH	DELHI		18

IMPORTANT TERMINOLOGIES

- **Attribute:** Attributes are the properties that define a relation. e.g.; **ROLL_NO**, **NAME**
- **Relation Schema:** A relation schema represents name of the relation with its attributes. e.g.; STUDENT (ROLL_NO, NAME, ADDRESS, PHONE and AGE) is relation schema for STUDENT. If a schema has more than 1 relation, it is called Relational Schema.
- **Tuple:** Each row in the relation is known as tuple. The above relation contains 4 tuples, one of which is shown as:

1	RAM	DELHI	9455123451	18

- **Relation Instance:** The set of tuples of a relation at a particular instance of time is called as relation instance. Table 1 shows the relation instance of STUDENT at a particular time. It can change whenever there is insertion, deletion or updation in the database.
- **Degree:** The number of attributes in the relation is known as degree of the relation. The **STUDENT** relation defined above has degree 5.
- **Cardinality:** The number of tuples in a relation is known as cardinality. The **STUDENT** relation defined above has cardinality 4.
- **Column:** Column represents the set of values for a particular attribute. The column **ROLL_NO** is extracted from relation STUDENT.

ROLL_NO
1
2
3
4

- **NULL Values:** The value which is not known or unavailable is called NULL value. It is represented by blank space. e.g.; PHONE of STUDENT having ROLL_NO 4 is NULL.

Constraints in Relational Model

While designing Relational Model, we define some conditions which must hold for data present in database are called Constraints. These constraints are checked before performing any operation (insertion, deletion and updation) in database. If there is a violation in any of constrains, operation will fail.

Domain Constraints: These are attribute level constraints. An attribute can only take values which lie inside the domain range. e.g,; If a constrains AGE>0 is applied on STUDENT relation, inserting negative value of AGE will result in failure.

Key Integrity: Every relation in the database should have atleast one set of attributes which defines a tuple uniquely. Those set of attributes is called key. e.g.; ROLL_NO in STUDENT is a key. No two students can have same roll number. So a key has two properties:

- It should be unique for all tuples.
- It can't have NULL values.

Referential Integrity: When one attribute of a relation can only take values from other attribute of same relation or any other relation, it is called referential integrity. Let us suppose we have 2 relations

STUDENT

ROLL_NO	NAME	ADDRESS	PHONE	AGE	BRANCH_CODE
1	RAM	DELHI	9455123451	18	CS
2	RAMESH	GURGAON	9652431543	18	CS
3	SUJIT	ROHTAK	9156253131	20	ECE
4	SURESH	DELHI		18	IT

BRANCH

BRANCH_CODE	BRANCH_NAME
CS	COMPUTER SCIENCE
IT	INFORMATION TECHNOLOGY
ECE	ELECTRONICS AND COMMUNICATION ENGINEERING
CV	CIVIL ENGINEERING

BRANCH_CODE of STUDENT can only take the values which are present in BRANCH_CODE of BRANCH which is called referential integrity constraint. The relation which is referencing to other relation is called REFERENCING RELATION (STUDENT in this case) and the relation to which other relations refer is called REFERENCED RELATION (BRANCH in this case).

ANOMALIES
An anomaly is an irregularity, or something which deviates from the expected or normal state. When designing databases, we identify three types of anomalies: Insert, Update and Delete.

Insertion Anomaly in Referencing Relation:

We can't insert a row in REFERENCING RELATION if referencing attribute's value is not present in referenced attribute value. e.g.; Insertion of a student with BRANCH_CODE 'ME' in STUDENT relation will result in error because 'ME' is not present in BRANCH_CODE of BRANCH.

Deletion/ Updation Anomaly in Referenced Relation:

We can't delete or update a row from REFERENCED RELATION if value of REFERENCED ATTRIBUTE is used in value of REFERENCING ATTRIBUTE. e.g; if we try to delete tuple from BRANCH having BRANCH_CODE 'CS', it will result in error because 'CS' is referenced by BRANCH_CODE of STUDENT, but if we try to delete the row from BRANCH with BRANCH_CODE CV, it will be deleted as the value is not been used by referencing relation. It can be handled by following method:

ON DELETE CASCADE:

It will delete the tuples from REFERENCING RELATION if value used by REFERENCING ATTRIBUTE is deleted from REFERENCED RELATION. e.g;, if we delete a row from BRANCH with BRANCH_CODE 'CS', the rows in STUDENT relation with BRANCH_CODE CS (ROLL_NO 1 and 2 in this case) will be deleted.

ON UPDATE CASCADE:
It will update the REFERENCING ATTRIBUTE in REFERENCING RELATION if attribute value used by REFERENCING ATTRIBUTE is updated in REFERENCED RELATION. e.g;, if we update a row from BRANCH with BRANCH_CODE 'CS' to 'CSE', the rows in STUDENT relation with BRANCH_CODE CS (ROLL_NO 1 and 2 in this case) will be updated with BRANCH_CODE 'CSE'.

SUPER KEYS:

Any set of attributes that allows us to identify unique rows (tuples) in a given relation are known as super keys. Out of these super keys we can always choose a proper subset among these which can be used as a primary key. Such keys are known as Candidate keys. If there is a combination of two or more attributes which is being used as the primary key then we call it as a Composite key.

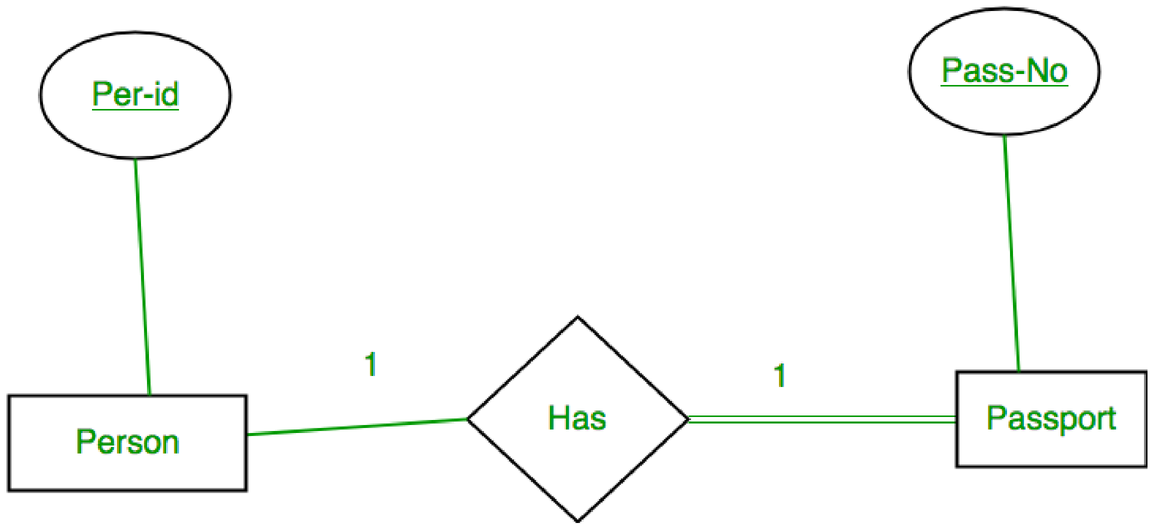
Conversion of ER to Relational Table

Conversion of ER Diagram to Relational model

- **1) Mapping regular/strong entity**
  For each strong entity set create a new relational independent table that includes all attributes as column. For composite attribute include only component attributes.
- **2) Mapping weak entity**
  Convert every weak entity set into a table where we take the discrimination attribute of the weak entity set and takes the primary key of the strong entity set as a foreign key and then declared the combination of discriminator attribute and foreign key as a primary key.
- **3) Mapping 1:1 Relationship**
  For a 1:1 Relationship between two entities **S** and **T**. Choose one of the relations, Example - **S** and include as foreign key in **S** the primary key of **T**. It is better to choose on entity total participation on **S** and include descriptive attribute.
- **4) Mapping 1:N Relationship**
  For 1:N relationship identify the entity **S** on **N** side of the relationship. Include a foreign key in **S** the Primary key of relation **T** also include Discipline attributes of 1:N attribute of **S**.
- **5) Mapping N:N Relationship**
  For each M:N relationship create a new relational table include in the new relation ,the primary key of the participating entities as a well as descriptive attributes. The primary key of the table will be the combination of primary keys of participating entities.
- **6) Mapping Multi-valued attribute**
  For every multi-valued attribute will make a new table where we will take primary key of main table as a foreign key and multi-valued attribute as a primary key.

Designing the ER diagram of system, we need to convert it to Relational models which can directly be implemented by any RDBMS like Oracle, MySQL etc. In this article we will discuss how to convert ER diagram to Relational Model for different scenarios.

Case 1: Binary Relationship with 1:1 cardinality with total participation of an entity



A person has 0 or 1 passport number and Passport is always owned by 1 person. So it is 1:1 cardinality with full participation constraint from Passport.

First Convert each entity and relationship to tables. Person table corresponds to Person Entity with key as Per-Id. Similarly Passport table corresponds to Passport Entity with key as Pass-No. HashTable represents relationship between Person and Passport (Which person has which passport). So it will take attribute Per-Id from Person and Pass-No from Passport.

Person			**Has**			**Passport**	
Per-Id	**Other Person Attribute**		**Per-Id**	**Pass-No**		**Pass-No**	**Other PassportAttribute**
PR1	–		PR1	PS1		PS1	–
PR2	–		PR2	PS2		PS2	–
PR3	–						

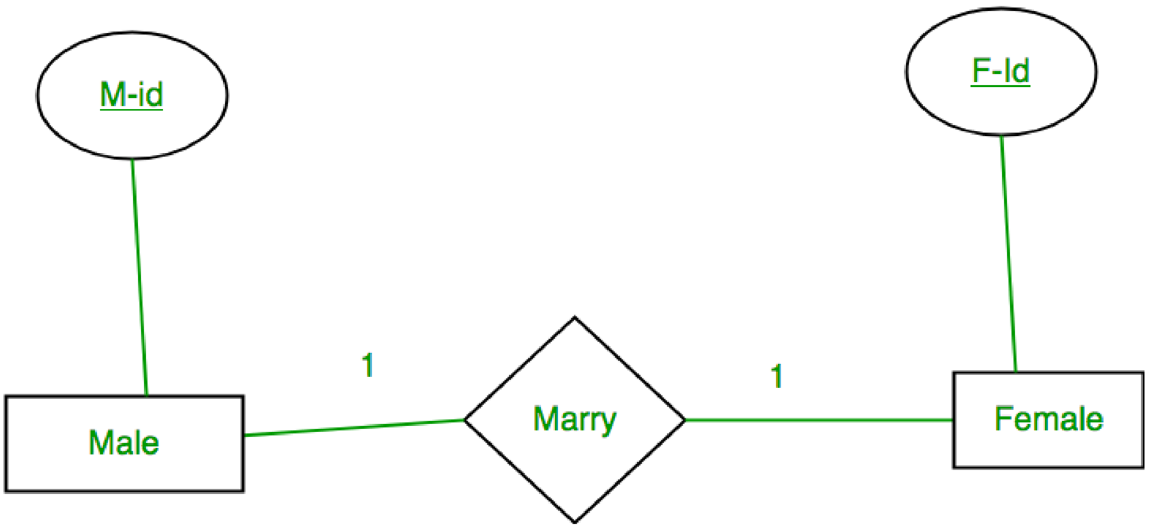
Table 1

As we can see from Table 1, each Per-Id and Pass-No has only one entry in **Hash**table. So we can merge all three tables into 1 with attributes shown in Table 2. Each Per-Id will be unique and not null. So it will be the key. Pass-No can't be key because for some person, it can be NULL.

Per-Id	Other Person Attribute	Pass-No	Other PassportAttribute

Table 2

Case 2: Binary Relationship with 1:1 cardinality and partial participation of both entities



A male marries 0 or 1 female and vice versa as well. So it is 1:1 cardinality with partial participation constraint from both. First Convert each entity and relationship to tables. Male table corresponds to Male Entity with key as M-Id. Similarly Female table corresponds to Female Entity with key as F-Id. Marry Table represents relationship between Male and Female (Which Male marries which female). So it will take attribute M-Id from Male and F-Id from Female.

Male			**Marry**			**Female**	
M-Id	Other Male Attribute		M-Id	F-Id		F-Id	Other FemaleAttribute
M1	–		M1	F2		F1	–
M2	–		M2	F1		F2	–
M3	–					F3	–

Table 3

As we can see from Table 3, some males and some females do not marry. If we merge 3 tables into 1, for some M-Id, F-Id will be NULL. So there is no attribute which is always not NULL. So we can't merge all three tables into 1. We can convert into 2 tables. In table 4, M-Id who are married will have F-Id associated. For others, it will be NULL. Table 5 will have information of all females. Primary Keys have been underlined.

M-Id	Other Male Attribute	F-Id

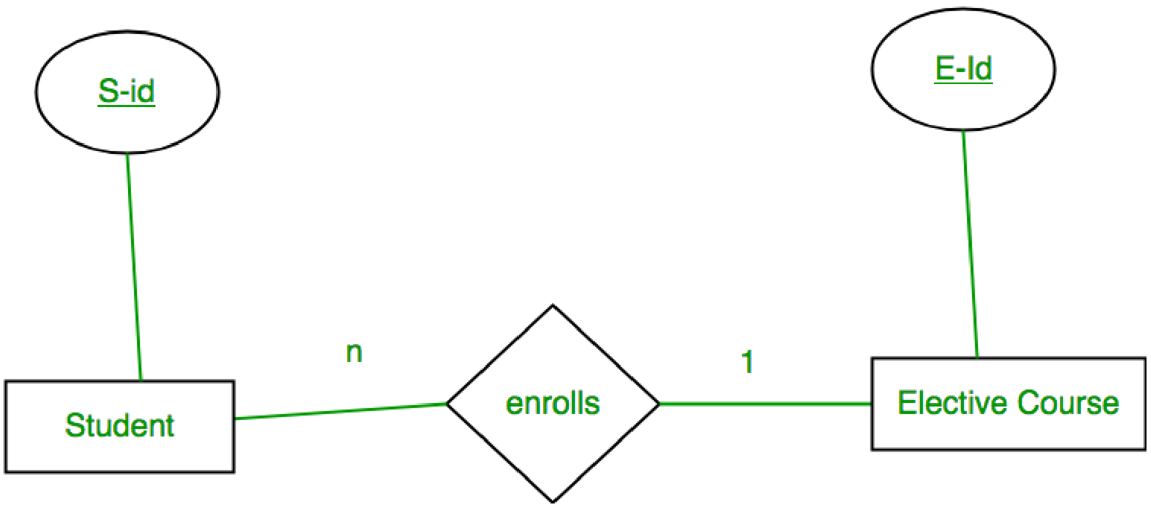
Table 4

F-Id	Other FemaleAttribute

Table 5

Note: Binary relationship with 1:1 cardinality will have 2 table if partial participation of both entities in the relationship. If atleast 1 entity has total participation, number of tables required will be 1.

Case 3: Binary Relationship with n: 1 cardinality



In this scenario, every student can enroll only in one elective course but for an elective course there can be more than one student. First Convert each entity and relationship to tables. Student table corresponds to Student Entity with key as S-Id. Similarly Elective_Course table corresponds to Elective_Course Entity with key as E-Id. Enrolls Table represents relationship between Student and Elective_Course (Which student enrolls in which course). So it will take attribute S-Id from and

Student E-Id from Elective_Course.

Student	
<u>S-Id</u>	Other Student Attribute
S1	–
S2	–
S3	–
S4	–

Enrolls	
<u>S-Id</u>	E-Id
S1	E1
S2	E2
S3	E1
S4	E1

Elective_Course	
<u>E-Id</u>	Other Elective CourseAttribute
E1	–
E2	–
E3	–

Table 6

As we can see from Table 6, S-Id is not repeating in Enrolls Table. So it can be considered as a key of Enrolls table. Both Student and Enrolls Table's key is same; we can merge it as a single table. The resultant tables are shown in Table 7 and Table 8. Primary Keys have been underlined.

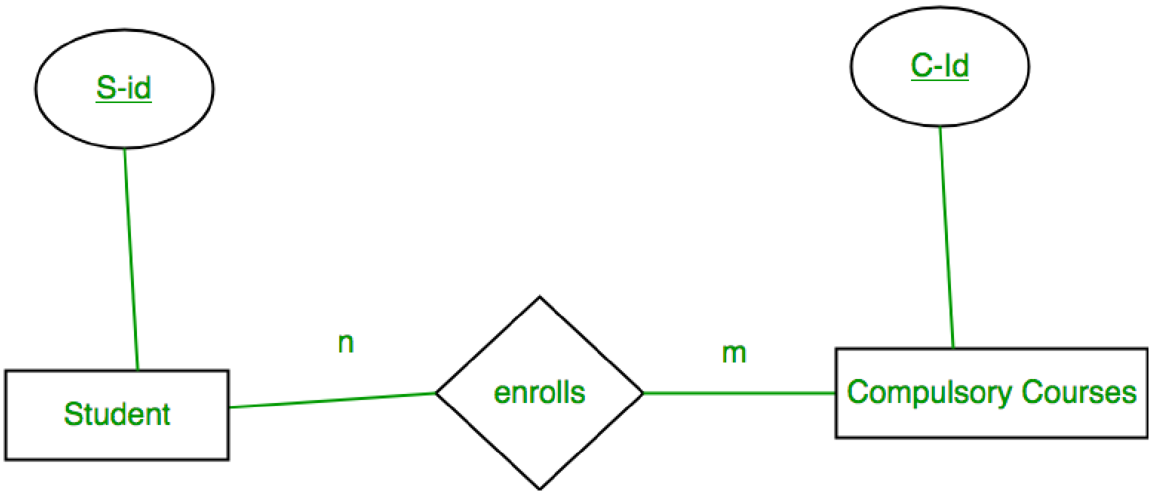
<u>S-Id</u>	Other Student Attribute	E-Id

Table 7

<u>E-Id</u>	Other Elective CourseAttribute

Table 8

Case 4: Binary Relationship with m: n cardinality



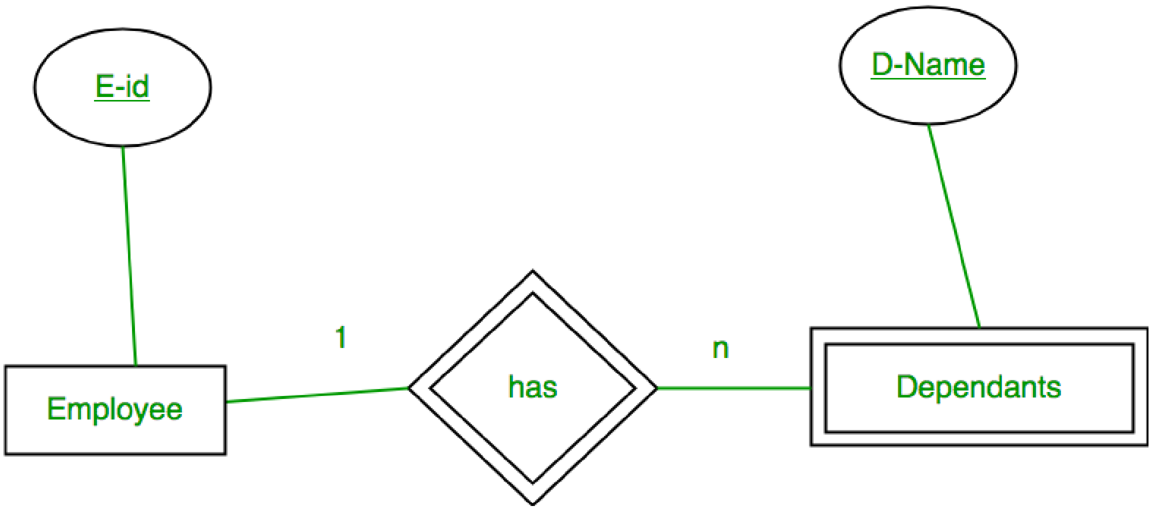
In this scenario, every student can enroll in more than 1 compulsory course and for a compulsory course there can be more than 1 student. First Convert each entity and relationship to tables. Student table corresponds to Student Entity with key as S-Id. Similarly Compulsory_Courses table corresponds to Compulsory Courses Entity with key as C-Id. Enrolls Table represents relationship between Student and Compulsory_Courses (Which student enrolls in which course). So it will take attribute S-Id from Person and C-Id from Compulsory_Courses.

Student			Enrolls			Compulsory_Courses	
S-Id	Other Student Attribute		S-Id	C-Id		C-Id	Other Compulsory CourseAttribute
S1	–		S1	C1		C1	–
S2	–		S1	C2		C2	–
S3	–		S3	C1		C3	–
S4	–		S4	C3		C4	–
			S4	C2			
			S3	C3			

Table 9

As we can see from Table 9, S-Id and C-Id both are repeating in Enrolls Table. But its combination is unique; so it can be considered as **Case** a key of Enrolls table. All tables' keys are different, these can't be merged. Primary Keys of all tables have been underlined.

5: Binary Relationship with weak entity



In this scenario, an employee can have many dependents and one dependent can depend on one employee. A dependent does not have any existence without an employee (e.g; you as a child can be dependent of your father in his company). So it will be a weak entity and its participation will always be total. Weak Entity does not have key of its own. So its key will be combination of key of its identifying entity (E-Id of Employee in this case) and its partial key (D-Name).
First Convert each entity and relationship to tables. Employee table corresponds to Employee Entity with key as E-Id. Similarly Dependents table corresponds to Dependent Entity with key as D-Name and E-Id. HashTable represents relationship between Employee and Dependents (Which employee has which dependents). So it will take attribute E-Id from Employee and D-

Name from Dependents.

Employee			Has			Dependents		
E-Id	Other Employee Attribute		E-Id	D-Name		D-Name	E-Id	Other DependentsAttribute
E1	–		E1	RAM		RAM	E1	–
E2	–		E1	SRINI		SRINI	E1	–
E3	–		E2	RAM		RAM	E2	–
			E3	ASHISH		ASHISH	E3	–

Table 10

As we can see from Table 10, E-Id, D-Name is key for **Has** as well as Dependents Table. So we can merge these two into 1. So the resultant tables are shown in Tables 11 and 12. Primary Keys of all tables have been underlined.

E-Id	Other Employee Attribute

Table 11

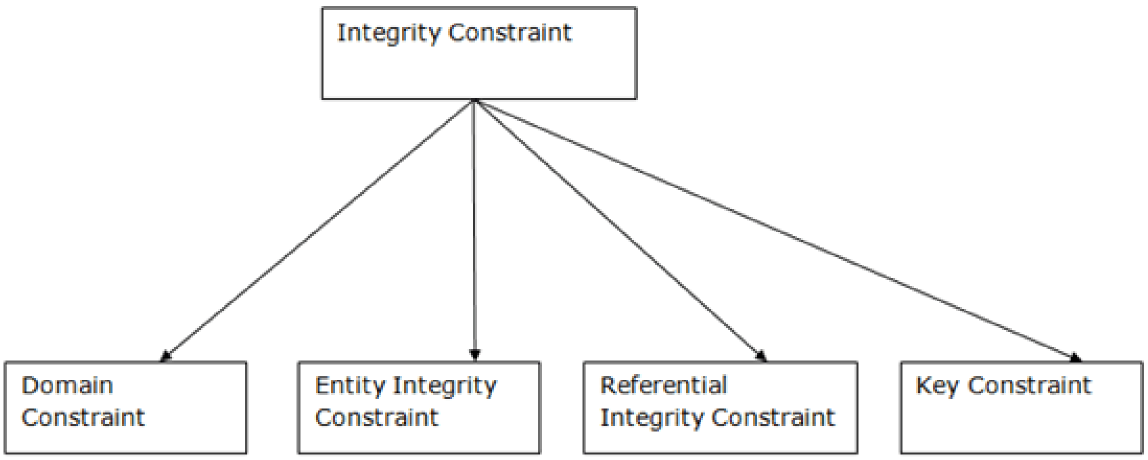
D-Name	E-Id	Other DependentsAttribute

Table 12

Integrity Constraints

- Integrity constraints are a set of rules. It is used to maintain the quality of information.
- Integrity constraints ensure that the data insertion, updating, and other processes have to be performed in such a way that data integrity is not affected.
- Thus, integrity constraint is used to guard against accidental damage to the database.

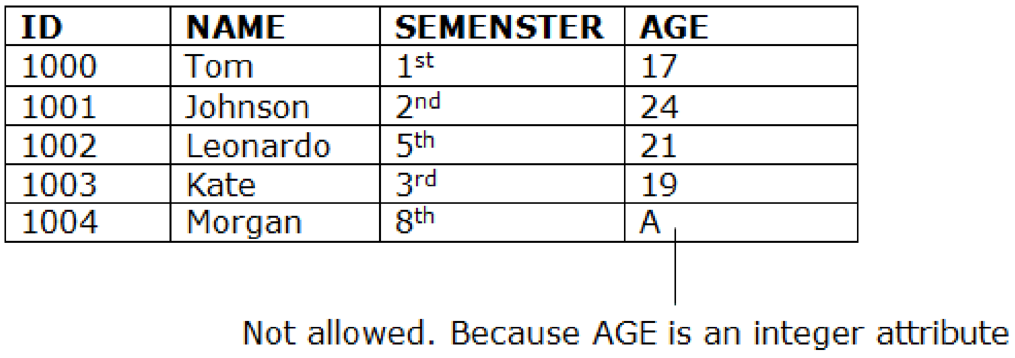
Types of Integrity Constraint



1. Domain constraints

- Domain constraints can be defined as the definition of a valid set of values for an attribute.
- The data type of domain includes string, character, integer, time, date, currency, etc. The value of the attribute must be available in the corresponding domain.

Example:



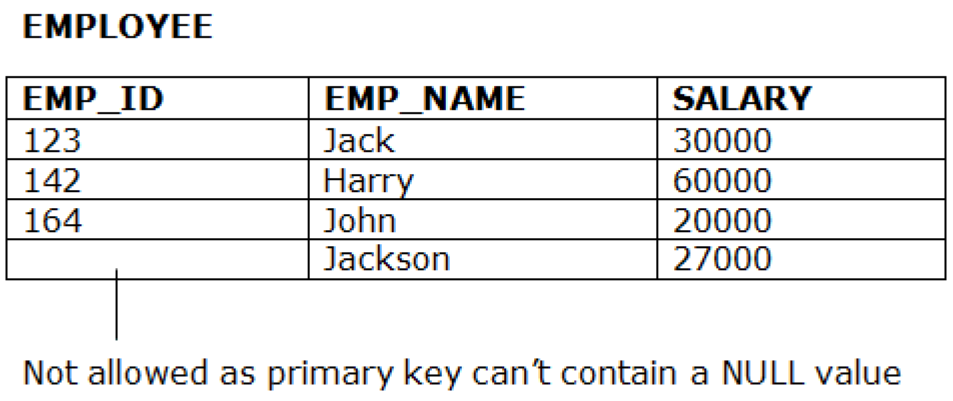
ID	NAME	SEMENSTER	AGE
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1004	Morgan	8th	A

Not allowed. Because AGE is an integer attribute

2. Entity integrity constraints

- The entity integrity constraint states that primary key value can't be null.
- This is because the primary key value is used to identify individual rows in relation and if the primary key has a null value, then we can't identify those rows.
- A table can contain a null value other than the primary key field.

Example:



EMPLOYEE

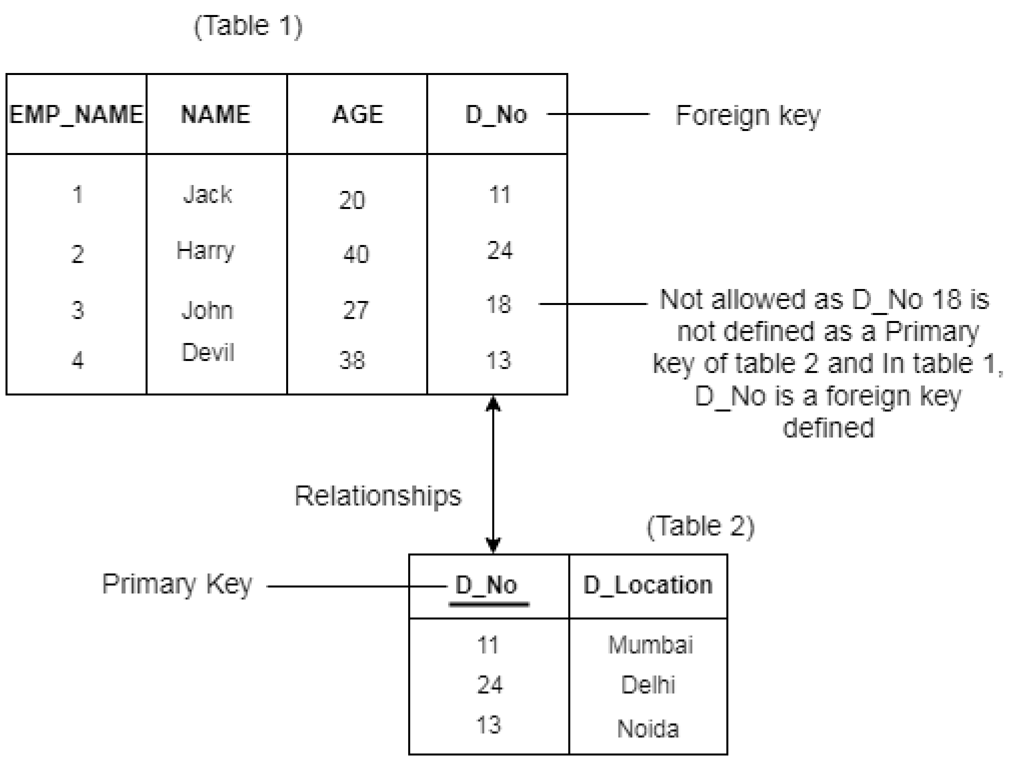
EMP_ID	EMP_NAME	SALARY
123	Jack	30000
142	Harry	60000
164	John	20000
	Jackson	27000

Not allowed as primary key can't contain a NULL value

3. Referential Integrity Constraints

- A referential integrity constraint is specified between two tables.
- In the Referential integrity constraints, if a foreign key in Table 1 refers to the Primary Key of Table 2, then every value of the Foreign Key in Table 1 must be null or be available in Table 2.

Example:


(Table 1)
Foreign key
Not allowed as D_No 18 is not defined as a Primary key of table 2 and In table 1, D_No is a foreign key defined
Relationships
(Table 2)
Primary Key

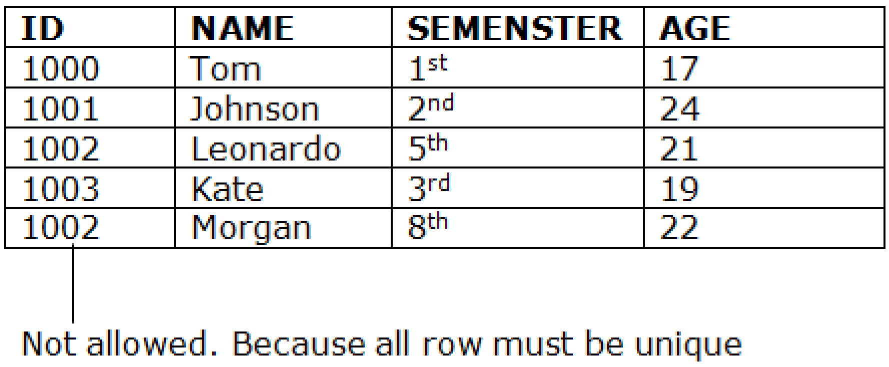
EMP_NAME	NAME	AGE	D_No
1	Jack	20	11
2	Harry	40	24
3	John	27	18
4	Devil	38	13

D_No	D_Location
11	Mumbai
24	Delhi
13	Noida

4. Key constraints

- Keys are the entity set that is used to identify an entity within its entity set uniquely.
- An entity set can have multiple keys, but out of which one key will be the primary key. A primary key can contain a unique and null value in the relational table.

Example:



ID	NAME	SEMENSTER	AGE
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1002	Morgan	8th	22

Not allowed. Because all row must be unique

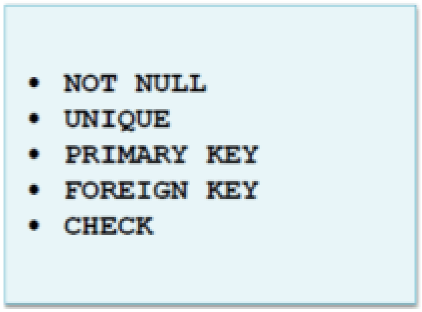
CONSTRAINTS

Constraints are the rules that are used to restrict the values in a database

- Constraints are used table-level rules to be applied forcibly.
- it prevents deletion of dependent data.
- it prevents incorrect data entry.
- They ensure that data is unique.
- They can be used at table or column level.

There are six types of integrity constraints in the Oracle database as follows.

1. Primary Key
2. Foreign Key
3. Unique Key
4. Index
5. Check
6. NOT Null



- NOT NULL
- UNIQUE
- PRIMARY KEY
- FOREIGN KEY
- CHECK

PRIMARY KEY CONSTRAINT

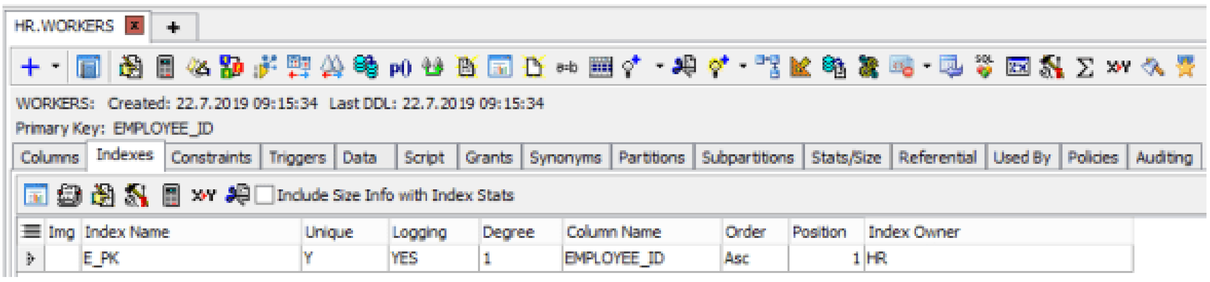
- A table can only have one primary key.
- Primary key column can not be Null.
- It also works as an index.

You can define primary key as follows.

```
CREATE TABLE HR.WORKERS
(
   EMPLOYEE_ID    NUMBER (6) CONSTRAINT E_PK PRIMARY KEY,
   FIRST_NAME     VARCHAR2 (20 BYTE),
   LAST_NAME      VARCHAR2 (25 BYTE),
   HIRE_DATE      DATE,
   JOB_ID         VARCHAR2 (10 BYTE),
   SALARY         NUMBER (8, 2)
)
```

- ‘Employee_id’ column is primary key, so can not get same values (duplicated records) and can not be null.
- Index is created on ‘Employee_id’ column automatically.

Primary key combines NOT NULL constraint and a unique constraint in a single declaration.

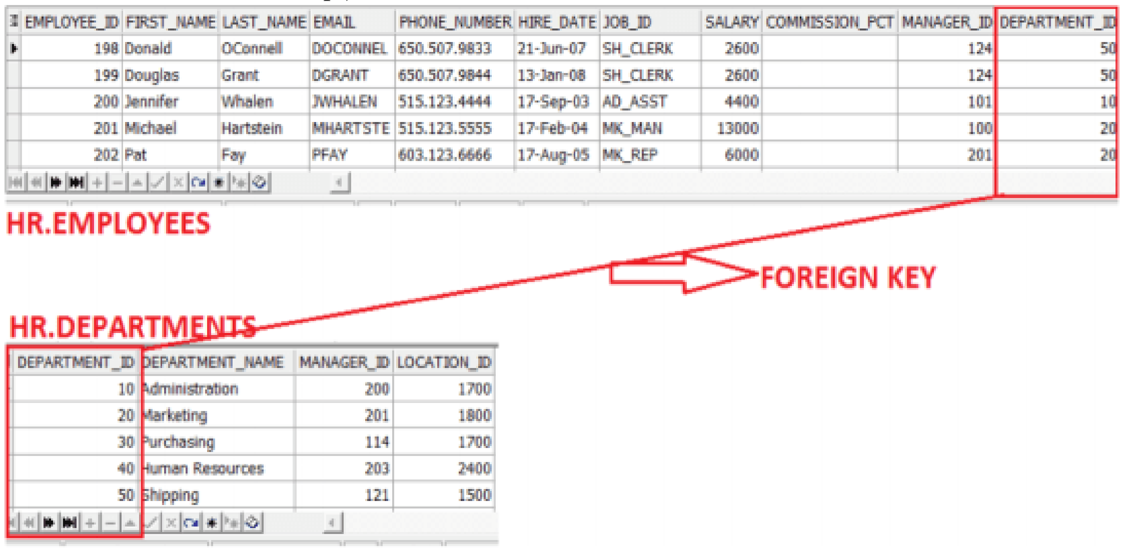


WORKERS: Created: 22.7.2019 09:15:34 Last DDL: 22.7.2019 09:15:34
Primary Key: EMPLOYEE_ID

Img	Index Name	Unique	Logging	Degree	Column Name	Order	Position	Index Owner
	E_PK	Y	YES	1	EMPLOYEE_ID	Asc	1	HR

FOREIGN KEY CONSTRAINT

- It guarantees that, the data in one column of a table must match from another table. (Parent – Child relationship)



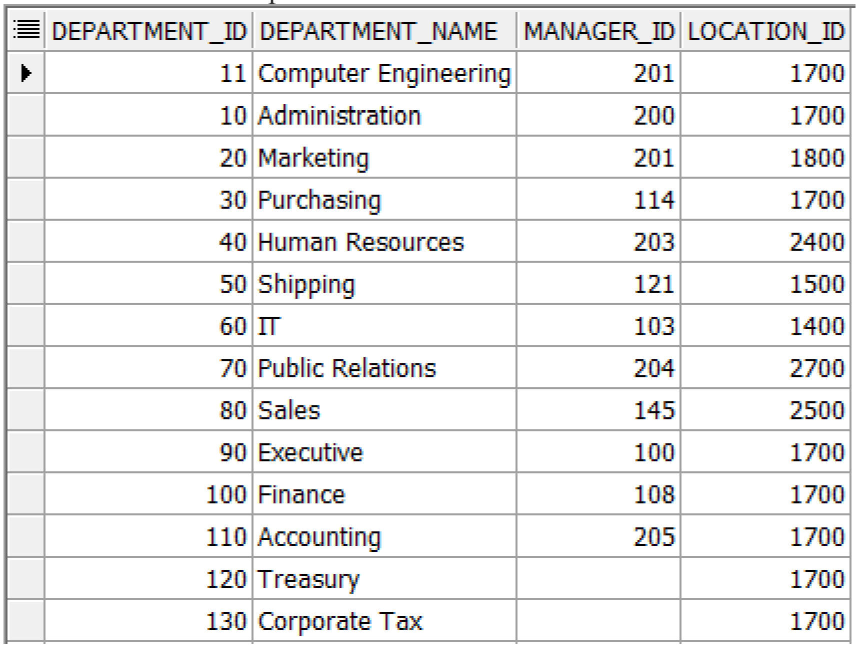
EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
198	Donald	OConnell	DOCONNEL	650.507.9833	21-Jun-07	SH_CLERK	2600		124	50
199	Douglas	Grant	DGRANT	650.507.9844	13-Jan-08	SH_CLERK	2600		124	50
200	Jennifer	Whalen	JWHALEN	515.123.4444	17-Sep-03	AD_ASST	4400		101	10
201	Michael	Hartstein	MHARTSTE	515.123.5555	17-Feb-04	MK_MAN	13000		100	20
202	Pat	Fay	PFAY	603.123.6666	17-Aug-05	MK_REP	6000		201	20

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing	114	1700
40	Human Resources	203	2400
50	Shipping	121	1500

- Let’s create a table with foreign key constraint.

```
CREATE TABLE HR.WORKERS
(
   EMPLOYEE_ID    NUMBER (6) CONSTRAINT E_PK PRIMARY KEY,
   FIRST_NAME      VARCHAR2 (20 BYTE),
   LAST_NAME       VARCHAR2 (25 BYTE),
   HIRE_DATE       DATE,
   JOB_ID          VARCHAR2 (10 BYTE),
   SALARY          NUMBER (8, 2),
   DEPARTMENT_ID  NUMBER (4),
   CONSTRAINT D_FK
   FOREIGN KEY (DEPARTMENT_ID)
   REFERENCES HR.DEPARTMENTS(department_id)
);
```

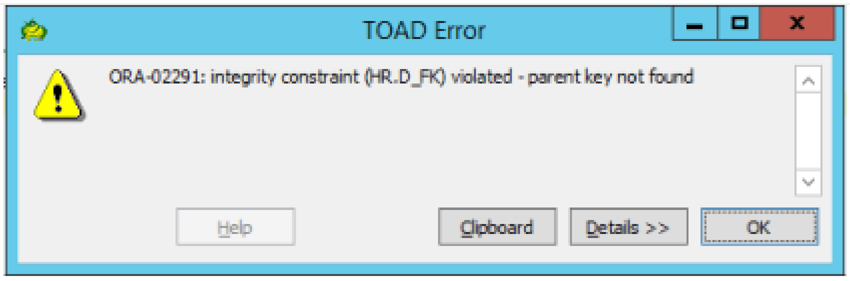
- Here is the Departments table's data.



DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
11	Computer Engineering	201	1700
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing	114	1700
40	Human Resources	203	2400
50	Shipping	121	1500
60	IT	103	1400
70	Public Relations	204	2700
80	Sales	145	2500
90	Executive	100	1700
100	Finance	108	1700
110	Accounting	205	1700
120	Treasury		1700
130	Corporate Tax		1700

- I will try to insert '123' to department_id column. I will get an error like below. Because we have foreign key constraint in department_id columnd and '123' doesn't exist in the Departments table.
- For example; I can insert '70' to department_id because it exists on the other table.

```
INSERT INTO hr.workers (employee_id, department_id)
   VALUES (1, 123);
```



ORA-02291: integrity constraint (HR.D_FK) violated - parent key not found

NOT NULL CONSTRAINT

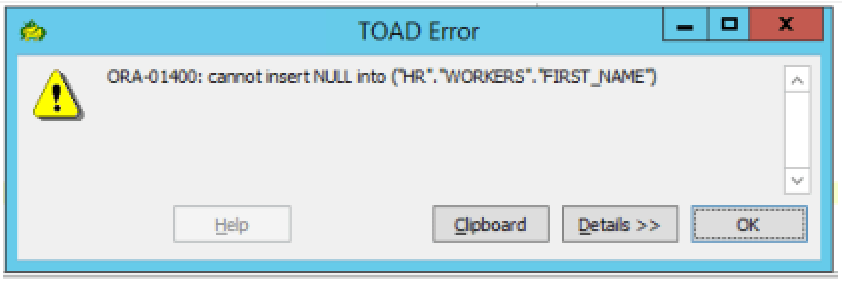
- Prevents specified columns value from being null.

You can define it as follows.

```
CREATE TABLE HR.WORKERS
(
   EMPLOYEE_ID   NUMBER (6),
   FIRST_NAME    VARCHAR2 (20 BYTE) NOT NULL,
   LAST_NAME     VARCHAR2 (25 BYTE),
   HIRE_DATE     DATE,
   JOB_ID        VARCHAR2 (10 BYTE),
   SALARY        NUMBER (8, 2)
);
```

- Let’s try to insert null value to ‘first_name’ column.

```
INSERT INTO hr.workers (first_name)
    VALUES ('');
```



```
ORA-01400: cannot insert NULL into ("HR"."WORKERS"."FIRST_NAME")
```

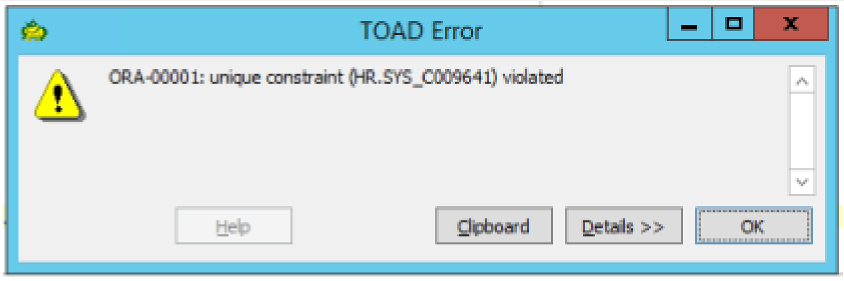
UNIQUE CONSTRAINT

- This Constraint type prevents multiple rows from having the same value in the same column. Namely it prevents duplication of data at columns.
- Unique Constraint can be created for one or more columns.
- More than one UC can be placed in a table.,

```
CREATE TABLE HR.WORKERS
(
   EMPLOYEE_ID   NUMBER (6) UNIQUE,
   FIRST_NAME    VARCHAR2 (20 BYTE),
   LAST_NAME     VARCHAR2 (25 BYTE),
   HIRE_DATE     DATE,
   JOB_ID        VARCHAR2 (10 BYTE),
   SALARY        NUMBER (8, 2)
)
```

- Let’s run this insert command two times and see the results. It will throw error at the second time we run. Because ‘employee_id’ colums must be unique.

```
INSERT INTO hr.workers (employee_id)
    VALUES (1);
```



```
ORA-00001: unique constraint (HR.SYS_C009641) violated
```

CHECK CONSTRAINT

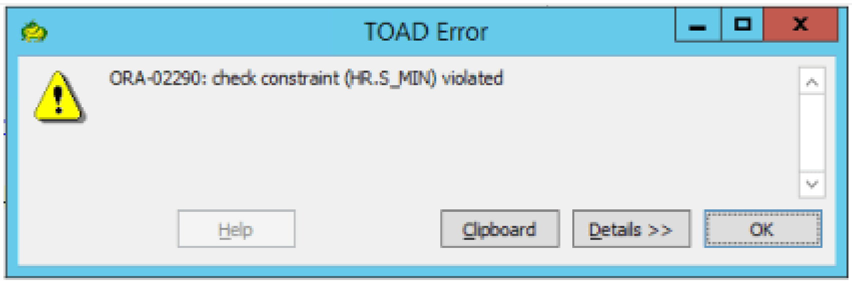
- it is used to a value in the database to comply with a specified condition.
- It defines the requirements for each row.
- CURRVAL, NEXTVAL, LEVEL and ROWNUM aliases can not be used.
- SYSDATE, UID, USER and USERNV functions can not be called.

- Queries referring to other values in other rows can not be performed.

```
CREATE TABLE HR.WORKERS
(
   EMPLOYEE_ID    NUMBER (6) CONSTRAINT E_PK PRIMARY KEY,
   FIRST_NAME     VARCHAR2 (20 BYTE),
   LAST_NAME      VARCHAR2 (25 BYTE),
   HIRE_DATE      DATE,
   JOB_ID         VARCHAR2 (10 BYTE),
   SALARY         NUMBER (8, 2) CONSTRAINT S_MIN CHECK (SALARY > 3000)
);
```

- We can't insert '2500' to salary column because of the check constraint.I can only insert values that higher than '3000'.

```
INSERT INTO hr.workers (employee_id, salary)
    VALUES (1, 2500);
```



ORA-02290: check constraint (HR.S_MIN) violated

Transaction Control Commands

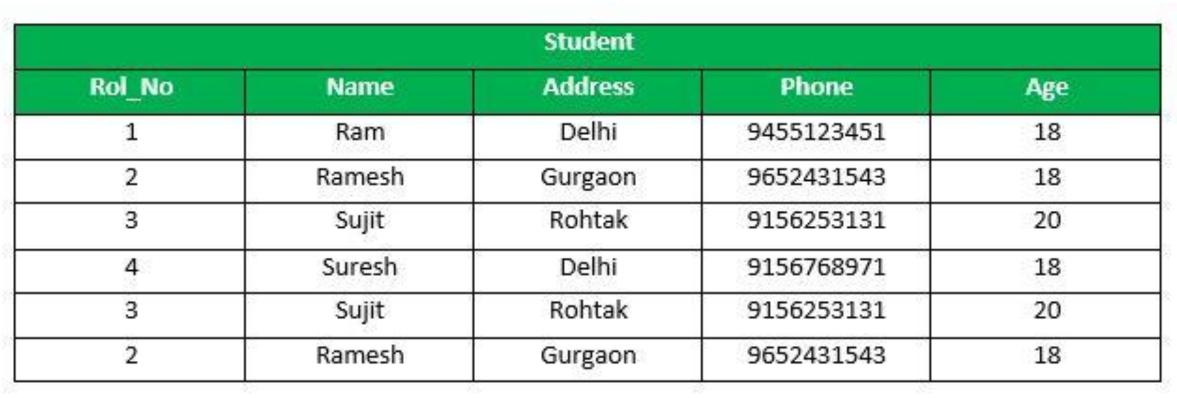
Transactions group a set of tasks into a single execution unit. Each transaction begins with a specific task and ends when all the tasks in the group successfully complete. If any of the tasks fail, the transaction fails. Therefore, a transaction has only two results: **success** or **failure**.
Incomplete steps result in the failure of the transaction. A database transaction, by definition, must be atomic, consistent, isolated and durable. These are popularly known as ACID properties.

How to implement Transactions using SQL?
Following commands are used to control transactions. It is important to note that these statements cannot be used while creating tables and are only used with the DML Commands such as – INSERT, UPDATE and DELETE.

1. BEGIN TRANSACTION: It indicates the start point of an explicit or local transaction.
Syntax:
BEGIN TRANSACTION transaction_name ;
2. SET TRANSACTION: Places a name on a transaction.
Syntax:
SET TRANSACTION [READ WRITE | READ ONLY];
3. COMMIT: If everything is in order with all statements within a single transaction, all changes are recorded together in the database is called **committed**. The COMMIT command saves all the transactions to the database since the last COMMIT or ROLLBACK command.

Syntax:

COMMIT;
Example: Sample table 1

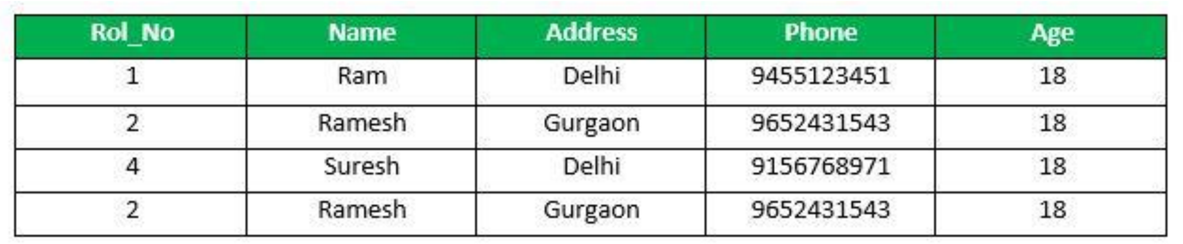


Student				
Rol_No	Name	Address	Phone	Age
1	Ram	Delhi	9455123451	18
2	Ramesh	Gurgaon	9652431543	18
3	Sujit	Rohtak	9156253131	20
4	Suresh	Delhi	9156768971	18
3	Sujit	Rohtak	9156253131	20
2	Ramesh	Gurgaon	9652431543	18

Following is an example which would delete those records from the table which have age = 20 and then COMMIT the changes in the database.
Queries:

DELETE FROM Student WHERE AGE = 20;
COMMIT;

Output:
Thus, two rows from the table would be deleted and the SELECT statement would look like,



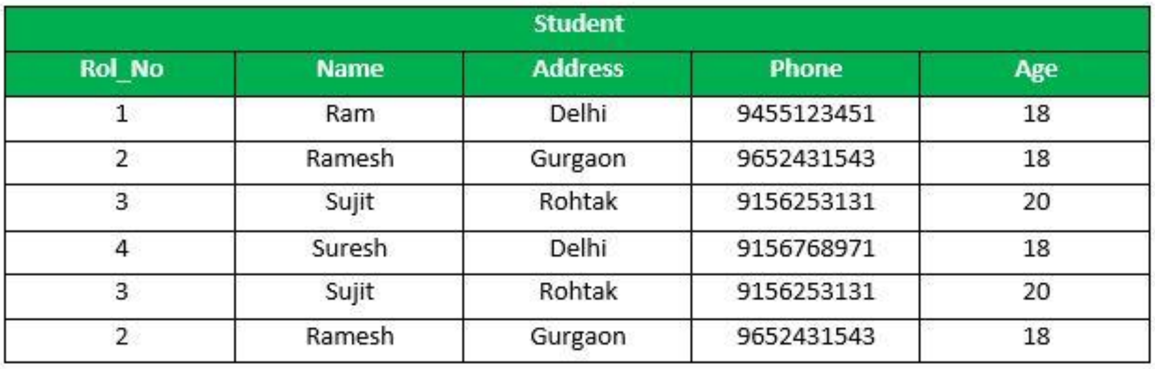
Rol_No	Name	Address	Phone	Age
1	Ram	Delhi	9455123451	18
2	Ramesh	Gurgaon	9652431543	18
4	Suresh	Delhi	9156768971	18
2	Ramesh	Gurgaon	9652431543	18

4. ROLLBACK: If any error occurs with any of the SQL grouped statements, all changes need to be aborted. The process of reversing changes is called **rollback**. This command can only be used to undo transactions since the last COMMIT or ROLLBACK command was issued.
Syntax:

ROLLBACK;

Example:
From the above example **Sample table1**,
Delete those records from the table which have age = 20 and then ROLLBACK the changes in the database.
Queries:

DELETE FROM Student WHERE AGE = 20;
ROLLBACK;
Output:



Student				
Rol_No	Name	Address	Phone	Age
1	Ram	Delhi	9455123451	18
2	Ramesh	Gurgaon	9652431543	18
3	Sujit	Rohtak	9156253131	20
4	Suresh	Delhi	9156768971	18
3	Sujit	Rohtak	9156253131	20
2	Ramesh	Gurgaon	9652431543	18

5. SAVEPOINT: creates points within the groups of transactions in which to ROLLBACK.
A SAVEPOINT is a point in a transaction in which you can roll the transaction back to a certain point without rolling back the entire transaction.

Syntax for Savepoint command:

```
SAVEPOINT SAVEPOINT_NAME;
```

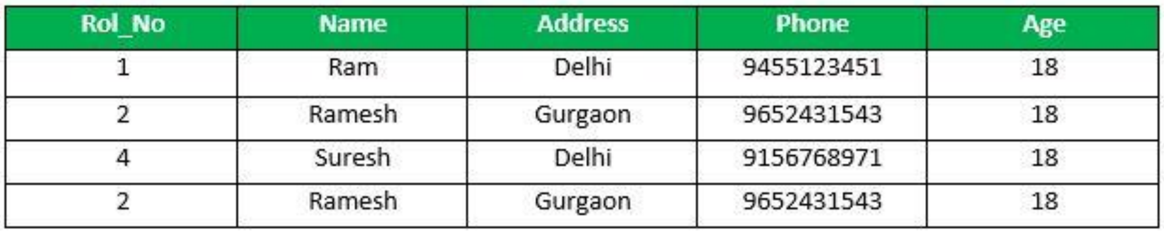
This command is used only in the creation of SAVEPOINT among all the transactions.
In general ROLLBACK is used to undo a group of transactions.
Syntax for rolling back to Savepoint command:

```
ROLLBACK TO SAVEPOINT_NAME;
```

you can ROLLBACK to any SAVEPOINT at any time to return the appropriate data to its original state.
Example:
From the above example **Sample table1**,
Delete those records from the table which have age = 20 and then ROLLBACK the changes in the database by keeping Savepoints.
Queries:

```
SAVEPOINT SP1;
//Savepoint created.
DELETE FROM Student WHERE AGE = 20;
//deleted
SAVEPOINT SP2;
//Savepoint created.
```

Here SP1 is first SAVEPOINT created before deletion.In this example one deletion have taken place.
After deletion again SAVEPOINT SP2 is created.
Output:

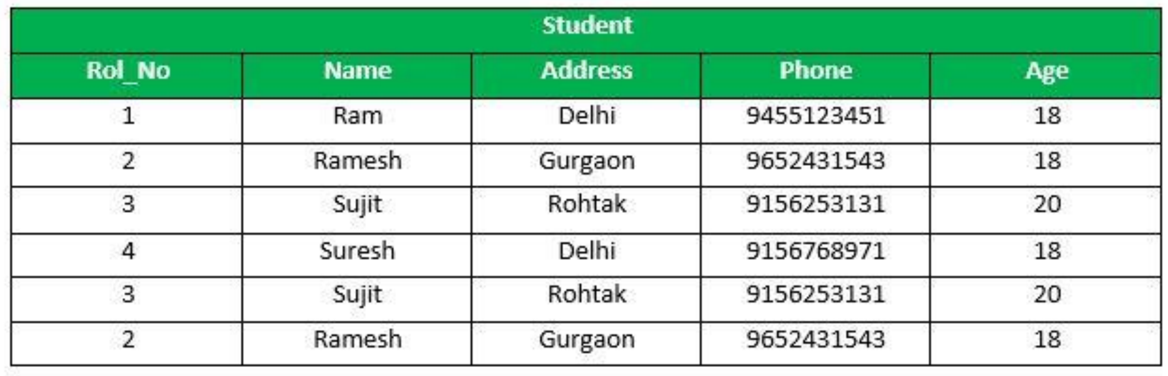


Rol_No	Name	Address	Phone	Age
1	Ram	Delhi	9455123451	18
2	Ramesh	Gurgaon	9652431543	18
4	Suresh	Delhi	9156768971	18
2	Ramesh	Gurgaon	9652431543	18

Deletion have been taken place, let us assume that you have changed your mind and decided to ROLLBACK to the SAVEPOINT that you identified as SP1 which is before deletion.

deletion is undone by this statement ,

ROLLBACK TO SP1;
//Rollback completed.



Student				
Rol_No	Name	Address	Phone	Age
1	Ram	Delhi	9455123451	18
2	Ramesh	Gurgaon	9652431543	18
3	Sujit	Rohtak	9156253131	20
4	Suresh	Delhi	9156768971	18
3	Sujit	Rohtak	9156253131	20
2	Ramesh	Gurgaon	9652431543	18

6. RELEASE SAVEPOINT:- This command is used to remove a SAVEPOINT that you have created.
Syntax:

RELEASE SAVEPOINT SAVEPOINT_NAME

Once a SAVEPOINT has been released, you can no longer use the ROLLBACK command to undo transactions performed since the last SAVEPOINT.

Relational Algebra

Relational Algebra is a procedural query language, it is used to provide a single table / relation as output of performing operations on more than one relations. Some of the basic relations will be discussed here.
In our course of learning, we will use three relations (table) −

Table 1: course

Course_id	Name
1	Computer science
2	Information Technology
3	mechanical

Table 2: students

Roll No.	Name	address	age
1	Ram	Delhi	18
2	Raju	hyderabad	20
4	Faiz	Delhi	22
5	Salman	hyderabad	20

Table 3: Hostel

St. No.	Name	address	age
1	Ram	Delhi	18
2	Akash	hyderabad	20
3	neha	Jhansi	21

On this relations, we will perform some operation to make new relation based on operations performed.

- **Selection operation (σ)** − The selection operator denoted by sigma σ is used to select the tuples of a relation based on some condition. Only those tuples that fall under certain conditions are selected.

Syntax

```
σ(condition)(relation_name)
```

Example

```
Select the student with course id 1.
σ(course_id = 1)(student)
```

Result

Roll No.	Name	address	age
4	Faiz	Delhi	22

- **Projection operation (∏)** The projection operator denoted by ∏ is used to select columns from a specific reaction. Only specific columns are selected.

Syntax

```
∏(column1 , column2 , … , columnn)(relation_name)
```

Example

```
Let’s select all students's name and no who are in hostel.
```

```
∏( st. No. , name)(hostel)
```

Result

St. No.	Name
1	Ram
2	Akash
3	neha

The row are always distinct in projection i.e. if their is any other student whose name is panjak the other one is removed.

- **Cross Product(X) -** Cross product is denoted using the X symbol and is used to find the value of join of two variables. In cross product each tuple of relation1 is crossed with each tuple of relation2. Which makes the output relation of the order nXm, where n is the number of tuples in relation1 and m is the number of tuples in relation2.

Syntax

```
relation1 X relation2
```

Example

Let's find cross product of course and hostel table.

```
student X course
```

St. No.	Name	address	age	Course_id	Name
1	Ram	Delhi	18	1	Computer science
1	Ram	Delhi	18	2	Information Technology
1	Ram	Delhi	18	3	mechanical
2	Akash	hyderabad	20	1	Computer science
2	Akash	hyderabad	20	2	Information Technology
2	Akash	hyderabad	20	3	mechanical
3	neha	Jhansi	21	1	Computer science

St. No.	Name	address	age	Course_id	Name
3	neha	Jhansi	21	2	Information Technology
3	neha	Jhansi	21	3	mechanical

- **Union (U) -** The union of two relations relation1 and relation2 will gives the tuples that are either in relation1 or in relation2 but tuples that are in both relation1 and relation2 are considered only once.

  Also both relations should be of the same domain for finding there union.

Syntax

```
relation1 U relation2
```

Example

Let’s find the union of student and hostel

```
student U hostel
```

Roll No.	Name	address	age
1	Ram	Delhi	18
2	Raju	hyderabad	20
4	Faiz	Delhi	22
5	Salman	hyderabad	20
2	Akash	hyderabad	20
3	neha	Jhansi	21

- **Minus (-) operator -** operator is denoted by - symbol. Relation1 - relation2 will result into a relation in which the tuple in relation1 and not in relation2 are present. For calculating minus too, the relations must be union compatible.

Syntax

```
relation1 - relation2
```

Example

Let’s find the operation student - hostel

```
student - hostel
```

Roll No.	Name	address	age
2	Raju	hyderabad	20
4	Faiz	Delhi	22
5	Salman	hyderabad	20

- **rename(ρ)** − the rename operation denoted by the ρ is used to rename the given relation to another name given.

Syntax

```
ρ(new_name , old_name)
```

Basic SQL Relational Algebra Operations
Relational Algebra devided in various groups

Unary Relational Operations

- SELECT (symbol: σ)
- PROJECT (symbol: π)
- RENAME (symbol: ρ)

Relational Algebra Operations From Set Theory

- UNION (υ)
- INTERSECTION ( ),
- DIFFERENCE (-)
- CARTESIAN PRODUCT (x)

Binary Relational Operations

- JOIN
- DIVISION

Let's study them in detail with solutions:

SELECT (σ)
The SELECT operation is used for selecting a subset of the tuples according to a given selection condition. Sigma(σ)Symbol denotes it. It is used as an expression to choose tuples which meet the selection condition. Select operator selects tuples that satisfy a given predicate.
$\sigma_p(r)$
σ is the predicate
r stands for relation which is the name of the table
p is prepositional logic
Example 1
$\sigma_{\text{topic = "Database"}}$ (Tutorials)
Output – Selects tuples from Tutorials where topic = 'Database'.
Example 2
$\sigma_{\text{topic = "Database" and author = "guru99"}}$(Tutorials)
Output – Selects tuples from Tutorials where the topic is 'Database' and 'author' is guru99.
Example 3
$\sigma_{\text{sales > 50000}}$ (Customers)

Output – Selects tuples from Customers where sales is greater than 50000

Projection(π)
The projection eliminates all attributes of the input relation but those mentioned in the projection list. The projection method defines a relation that contains a vertical subset of Relation.
This helps to extract the values of specified attributes to eliminates duplicate values. (pi) symbol is used to choose attributes from a relation. This operator helps you to keep specific columns from a relation and discards the other columns.
Example of Projection:
Consider the following table

CustomerID	**CustomerName**	**Status**
1	Google	Active
2	Amazon	Active
3	Apple	Inactive
4	Alibaba	Active

Here, the projection of CustomerName and status will give
$\Pi_{\text{CustomerName, Status}}$ (Customers)

CustomerName	**Status**
Google	Active
Amazon	Active
Apple	Inactive
Alibaba	Active

Rename (ρ)
Rename is a unary operation used for renaming attributes of a relation.
ρ (a/b)R will rename the attribute 'b' of relation by 'a'.

Union operation (υ)
UNION is symbolized by ∪ symbol. It includes all tuples that are in tables A or in B. It also eliminates duplicate tuples. So, set A UNION set B would be expressed as:
The result <- A ∪ B
For a union operation to be valid, the following conditions must hold –

- R and S must be the same number of attributes.
- Attribute domains need to be compatible.
- Duplicate tuples should be automatically removed.

Example
Consider the following tables.

Table A		**Table B**	
column 1	**column 2**	**column 1**	**column 2**
1	1	1	1
1	2	1	3

A ∪ B gives

Table A ∪ B

column 1	column 2
1	1
1	2
1	3

Set Difference (-)
– Symbol denotes it. The result of A – B, is a relation which includes all tuples that are in A but not in B.

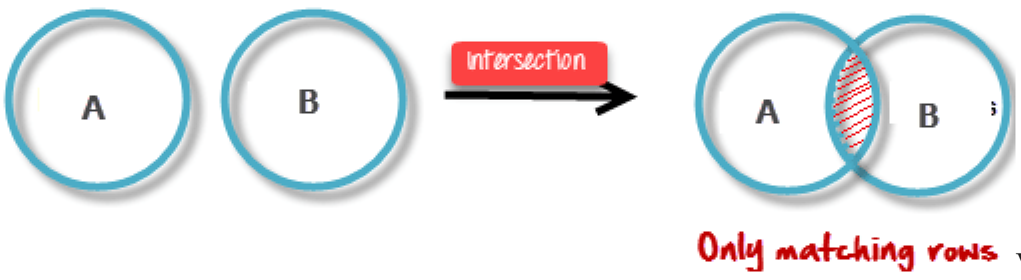
- The attribute name of A has to match with the attribute name in B.
- The two-operand relations A and B should be either compatible or Union compatible.
- It should be defined relation consisting of the tuples that are in relation A, but not in B.

Example
A-B

Table A – B

column 1	column 2
1	2

Intersection
An intersection is defined by the symbol ∩
A ∩ B
Defines a relation consisting of a set of all tuple that are in both A and B. However, A and B must be union-compatible.



Visual Definition of Intersection
Example:
A ∩ B

Table A ∩ B

column 1	column 2
1	1

Cartesian Product(X) in DBMS
Cartesian Product in DBMS is an operation used to merge columns from two relations. Generally, a cartesian product is never a meaningful operation when it performs alone. However, it becomes meaningful when it is followed by other operations. It is also called Cross Product or Cross Join.

Example – Cartesian product

$\sigma_{\text{column 2} = \text{'1'}}$ (A X B)

Output – The above example shows all rows from relation A and B whose column 2 has value 1

σ column 2 = ‘1’ (A X B)	
column 1	**column 2**
1	1
1	1

Join Operations

Join operation is essentially a cartesian product followed by a selection criterion.

Join operation denoted by ⋈.

JOIN operation also allows joining variously related tuples from different relations.

Types of JOIN:

Various forms of join operation are:

Inner Joins:

- Theta join
- EQUI join
- Natural join

Outer join:

- Left Outer Join
- Right Outer Join
- Full Outer Join

Inner Join:

In an inner join, only those tuples that satisfy the matching criteria are included, while the rest are excluded. Let’s study various types of Inner Joins:

Theta Join:

The general case of JOIN operation is called a Theta join. It is denoted by symbol **θ**

Example

$A \bowtie_{\theta} B$

Theta join can use any conditions in the selection criteria.

For example:

$A \bowtie_{\text{A.column 2} > \text{B.column 2}}$ (B)

A ⋈ A.column 2 > B.column 2 (B)	
column 1	**column 2**
1	2

EQUI join:

When a theta join uses only equivalence condition, it becomes a equi join.

For example:

$A \bowtie_{\text{A.column 2} = \text{B.column 2}}$ (B)

A ⋈ A.column 2 = B.column 2 (B)

column 1	**column 2**
1	1

EQUI join is the most difficult operations to implement efficiently using SQL in an RDBMS and one reason why RDBMS have essential performance problems.

NATURAL JOIN (⋈)
Natural join can only be performed if there is a common attribute (column) between the relations. The name and type of the attribute must be same.
Example
Consider the following two tables

C	
Num	**Square**
2	4
3	9

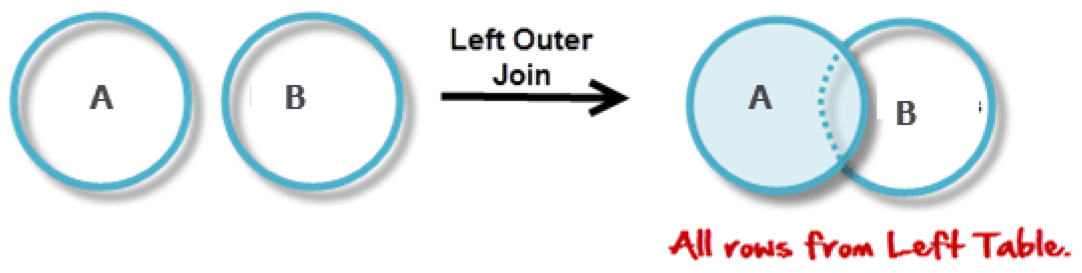
D	
Num	**Cube**
2	8
3	27

C ⋈ D

C ⋈ D		
Num	**Square**	**Cube**
2	4	8
3	9	27

OUTER JOIN
In an outer join, along with tuples that satisfy the matching criteria, we also include some or all tuples that do not match the criteria.

Left Outer Join(A ⟕ B)
In the left outer join, operation allows keeping all tuple in the left relation. However, if there is no matching tuple is found in right relation, then the attributes of right relation in the join result are filled with null values.



Consider the following 2 Tables

A

Num	**Square**
2	4
3	9
4	16

B

Num	**Cube**
2	8
3	18
5	75

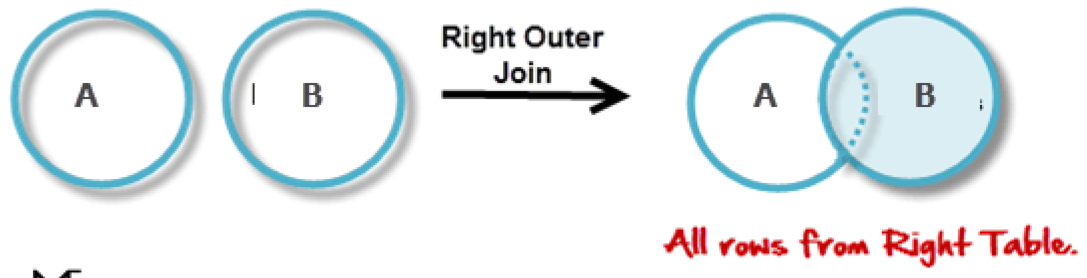
A ⟕ B

A ⋈ B

Num	**Square**	**Cube**
2	4	8
3	9	18
4	16	–

Right Outer Join: (A ⟖ B)

In the right outer join, operation allows keeping all tuple in the right relation. However, if there is no matching tuple is found in the left relation, then the attributes of the left relation in the join result are filled with null values.



A ⟖ B

A ⋈ B

Num	**Cube**	**Square**
2	8	4
3	18	9
5	75	–

Full Outer Join: (A ⟗ B)
In a full outer join, all tuples from both relations are included in the result, irrespective of the matching condition.
A ⟗ B

A ⋈ B

Num	**Cube**	**Square**
2	4	8
3	9	18
4	16	–
5	–	75