Dayananda Sagar College of Engineering

Kumara Swamy Layout,Bangalore-560078

Department of Artificial Intelligence & Machine Learning

Unit -3

Relational Data Model and Relational Database Constraints: Relational Model Concepts, Relational Model Constraints and Relational Database Schemas, Update Operations, Transactions and Dealing with Constraint Violations.

Relational Algebra: Unary Relational Operations, SELECT, and PROJECT, Relational Algebra Operations from Set Theory Binary Relational Operations: JOIN and DIVISION, Additional Relational Operations, Examples of Queries in Relational Algebra.

Text Book:

1. Database systems Models, Languages, Design and Application Programming, RamezElmasri and Shamkant B. Navathe, 7th Edition, 2017, Pearson.

2. Database management systems, Ramakrishnan, and Gehrke, 3rd Edition, 2014, McGraw Hill

Prepared by,
Dr.Aruna M G
Associate Professor
Department of AI&ML
DSCE
Bangalore

Both Relational Model and Relational Algebra

Relational Model Concepts

- Relational Model: The model was first proposed by Dr. E.F. Codd or Ted Codd of IBM in 1970 .
- The relational model uses the concepts of a mathematical relation-which looks like a table of values. Now in several commercial products (DB2, ORACLE, SQL Server, SYBASE, INFORMIX).
- The table is called a relation, a row is called a tuple, and column header is called attribute.

1. Relational Model

- It represents database as a collection of relations.
- A relation (table) contains rows and columns.
- Each column represents one type of information.
- Each row represents related data.

2. Relation

- The main construct for representing data in relational model is called relation.
- In relational model terminology, the table is called a relation.
- When a relation is thought of as a table of values, each row in the table represents a collection of related data values.
- The table name and column names are used to help in interpreting the meaning of the values in each row.
- A relation may be thought of as a set of rows and set of columns.
- Each row represents a fact that corresponds to a real-world entity or relationship.
- Each row has a value of an item or set of items that uniquely identifies that row in the table.
- Sometimes row-ids or sequential numbers are assigned to identify the rows in the table.
- Each column typically is called by its column name or column header or attribute name.

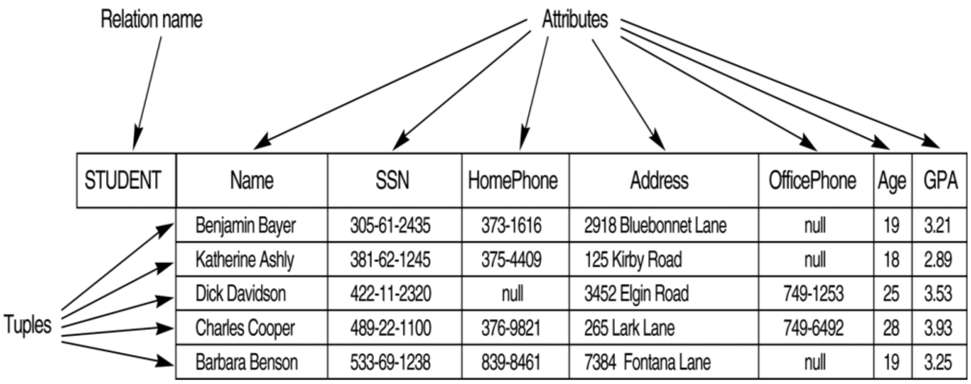
USN	NAME	SEMESTER	MARKS
1DS21AI001	Ajay	5	78.8
1DS20AI045	Sanujana	7	67.4
1DS22AI010	Gupta	3	68.3
1DS20AI012	Akash	7	81.3

3. Attribute

- In relational model terminology, column of a table is represented as a attribute.
- It is also known as field.

4. Tuple

- In relational model terminology, row of a table is represented as a tuple.
- It is also known as record.



STUDENT	Name	SSN	HomePhone	Address	OfficePhone	Age	GPA
	Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	null	19	3.21
	Katherine Ashly	381-62-1245	375-4409	125 Kirby Road	null	18	2.89
	Dick Davidson	422-11-2320	null	3452 Elgin Road	749-1253	25	3.53
	Charles Cooper	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
	Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	null	19	3.25

5. *Domain*

- A domain D is a set of atomic values.
- Atomic means values are distinct & indivisible.
- A domain is given a name, data type & format.

  Example:

Domain Name	**Datatype**	**Format**
DoJ	date	mm-dd-yyyy
Phno	String	ddd-ddd-dddddd

6. Relational Schema

- A relational schema R, denoted by R(A1,…An) is made up of relation name R & list of attributes A1,A2…An.
- Each attribute A_i is the name of a role played by some domain D in the relation schema R.
- D is called the domain of A_i & denoted by dom(A_i).
- A relation schema is used to describe a relation: R is called name of the relation.

7. Degree of Relation

- The total number of attributes in a relation is called degree (or arity) of relation.

  Example:
- STUDENT(Name, SSN, HomePhone, Address, OfficePhone, Age, GRA)

  No. of Attribute = 7

  The degree of relation STUDENT is 7

- The domain for the attributes of the STUDENT are dom(Name)=Names

dom(SSN)=Social_Security_Number

8. Cardinality :

The cardinality of a relation instance is the number of tuples in it. The cardinality of relation STUDENT is 5

9. Relation (Relation State)

- A relation (or relation state) r of the relation schema R(A1,..An),also denoted by r(R) is a set of n tuples r={t1,t2,…,tn}.
- Each n-tuple t is an ordered list of n values t=<v1,..,vn> where each value v_i , 1<=i<=n, is an element of dom(Ai) or a special null value.

  Definition of relation can be restated as follows:
- A relation (or relation state) r(R) is a mathematical relation of degree n of the domains dom(A1),dom(A2),..dom(An),which is the subset of the cartesian product of the domains that define R:
- $R(A_1, A_2, ..., A_n)$

  $r(R) \subset dom\ (A_1)\ X\ dom\ (A_2)\ X\ ....X\ dom(A_n)$
- R: schema of the relation
- r of R: a specific "value" or population of R.
- R is also called the intension of a relation
- r is also called the extension of a relation

FORMAL DEFINITIONS

- Let S1 = {0,1}
- Let S2 = {a,b,c}
- Let R ⊂ S1 X S2
- Then for example: r(R) = {<0,a> , <0,b> , <1,c> }

is one possible “state” or “population” or “extension” r of the relation R, defined over domains S1 and S2. It has three tuples.

DEFINITION SUMMARY

Informal Terms	**Formal Terms**
Table	Relation
Column	Attribute/Domain
Row	Tuple
Values in a column	Domain
Table Definition	Schema of a Relation
Populated Table	Extension

10. Characteristics Of Relations

1. Ordering of tuples in a relation r(R):

- The tuples are not considered to be ordered, even though they appear to be in the tabular form. The relation does not require this ordering

2. Ordering of attributes in a relation schema R (and of values within each tuple):

- The ordering of attributes is not important, because the attribute name appears with its value.
- There is no reason to prefer having one attribute value appear before another in a tuple.
- The attributes in $R(A_1, A_2, ..., A_n)$ and the values in $t=<v_1, v_2, ..., v_n>$ to be ordered .

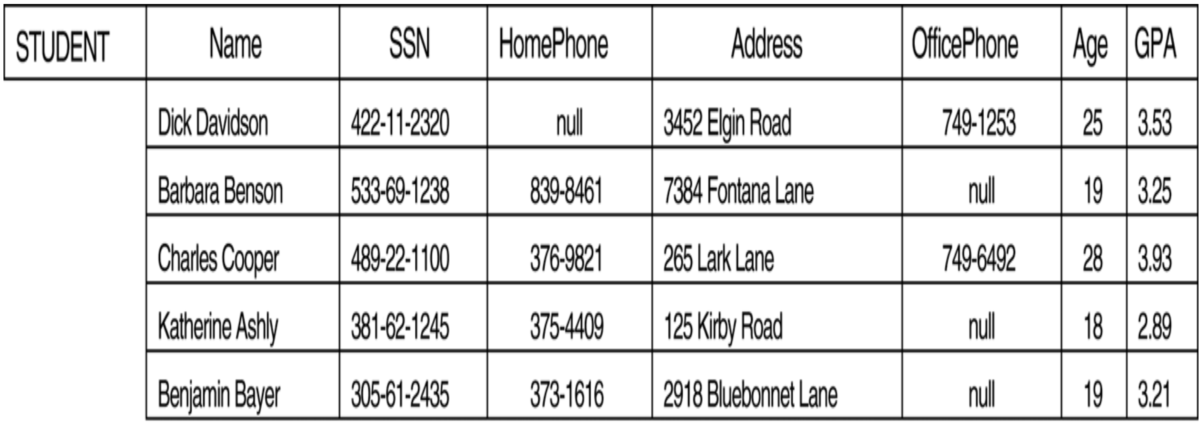
Ex: STUDENT("Regno","Name","Age","Sex")

Valid: t1=<"1SG04CS001","Aruna",23,"F")

Invalid: t2=<"aruna",23,"!SG04CS001","F")

3.Values in a tuple:

- All values are considered atomic(indivisible).
- A special null value is used to represent values that are unknown or inapplicable to certain tuples.
- In general, NULL values, means value unknown or value exists but is not available



STUDENT	Name	SSN	HomePhone	Address	OfficePhone	Age	GPA
	Dick Davidson	422-11-2320	null	3452 Elgin Road	749-1253	25	3.53
	Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	null	19	3.25
	Charles Cooper	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
	Katherine Ashly	381-62-1245	375-4409	125 Kirby Road	null	18	2.89
	Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	null	19	3.21

11. Relational Model Notation:

- A relational schema R of degree n is denoted by R(A_1 ,..,A_n).
- An n-tuple t in a relation r(R) is denoted by t=<v_1,.,v_n >,where v_i is the value corresponding to attribute A_i.
- The component values of a tuple t by t[A_i] = v_i (the value of attribute A_i for tuple t). Similarly, t[A_u, A_v, ..., A_w] refers to the subtuple of t containing the values <v_u,v_w,..,v_z>attributes A_u, A_v,., A_w, respectively.
- The letters Q,R,S denote relation names.
- The letters q,r,s denote relation states.
- The letters t,u,v denote tuples.
- STUDENT(SSN,Name,..) refers to the relation schema. The name of a relation schema indicates the current set of tuples in the relation.
- An attribute A can be qualified with the relation name by R.A- Eg.STUDENT.Name or STUDENT.Age.

Note:- the same name may be used for two attributes in different relations.

t[Name]=<'Barbara Benson'>
t[Name,SSN,Age]=<'Barbara','563-789',23>
t=<'Barbara Benson','563-789',23,null,3.45>

12. Relational Integrity Constraints

- Constraints are conditions that must hold on all valid relation instances.
- A constraints is a limitation or restrictions that can be placed on a field to ensure that the user enters only valid data.

Categories of constraints

1. Inherent model-based or implicit constraints
2. Schema-based or explicit constraints
3. Application-based or semantic constraints or business rules

1. Inherent model-based or implicit constraints

Constraints that is inherent in a data model.

Example: The constraint that a relation cannot have duplicate values

2. Schema-based or explicit constraints

This type of constraint can be expressed directly in schemas of the data model by specifying in DDL

- There are four main types of constraints:
  1. **Domain constraints**
  2. **Key constraints**
  3. **Entity integrity constraints**
  4. **Referential integrity constraints**

Domain Constraints

- Domain constraints specify that the value of each attribute A must be an atomic value from the domain dom(A) for that attribute.

Example : Domain is given a name, datatype and format.

Student_Name Varchar(30)

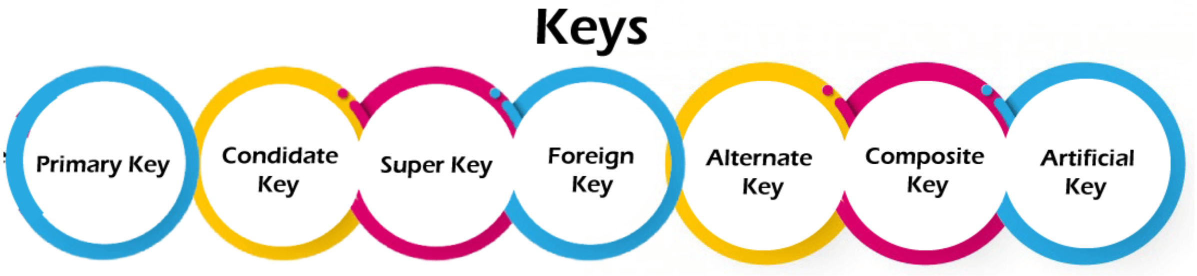
Datatype associated with domains typically include

- Integer (short-integer, integer, long-integer)
- Real Number (float and double-precision float)
- Character
- Fixed and Variable length string
- Date and Time
- Money

Domain Constraint = data type + Constraints (NOT NULL / UNIQUE / PRIMARY KEY / FOREIGN KEY / CHECK / DEFAULT)

Types of keys:

1. Super Key
2. Primary Key
3. Candidate Key
4. Alternate Key
5. Foreign Key
6. Compound Key
7. Composite Key
8. Surrogate Key/Artificial key



Key Constraints

- A relation is defined as a set of tuples.
- All the tuples in a relation must be distinct/Unique/no duplication.
- This means no two tuples can have the same combination of values for all their attributes.

- **Keys play an important role in the relational database**.
- It is used to uniquely identify any record or row of data from the table. It is also used to establish and identify relationships between tables.

A Key satisfies two Constraints:

1)Superkey of R:

A set of attributes SK of R such that no two tuples in any valid relation instance r(R) will have the same value for SK.

That is, for any distinct tuples t1 and t2 in r(R), t1[SK] ≠ t2[SK].

- A super key is a group of single or multiple keys that identifies rows or a tuple uniquely in a table.
- A super key is the superset of a key known as a **Candidate key**
- Adding zero or more attributes to the candidate key generates the super key.
- A candidate key is a super key but vice versa is not true.
- Super Key values may also be NULL.

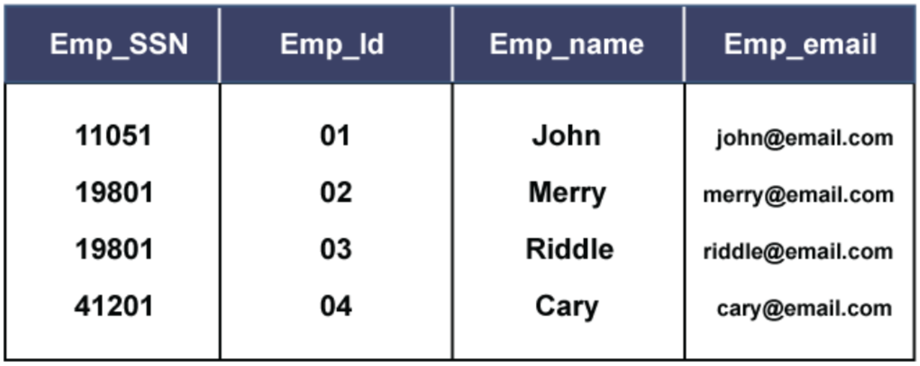
Example : Let's consider an EMPLOYEE_DETAIL **table example where we have the following attribute**:

Emp_SSN: The SSN number is stored in this field.

Emp_Id: An attribute that stores the value of the employee identification number.

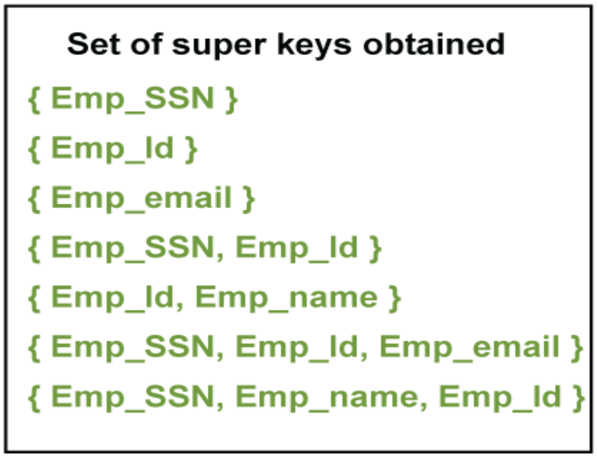
Emp_name: An attribute that stores the name of the employee holding the specified employee id.

Emp_email: An attribute that stores the email id of the specified employees.



Emp_SSN	Emp_Id	Emp_name	Emp_email
11051	01	John	john@email.com
19801	02	Merry	merry@email.com
19801	03	Riddle	riddle@email.com
41201	04	Cary	cary@email.com

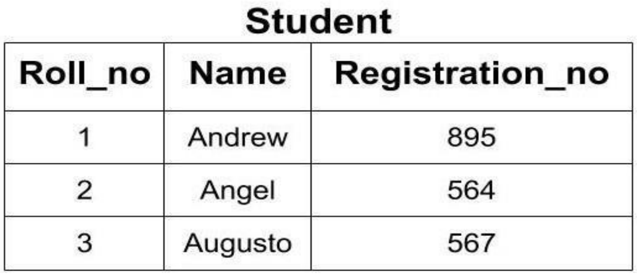
Note : These all are the set of super keys which, together or combining with other prime attributes, can identify a table uniquely.



Set of super keys obtained

{ Emp_SSN }
{ Emp_Id }
{ Emp_email }
{ Emp_SSN, Emp_Id }
{ Emp_Id, Emp_name }
{ Emp_SSN, Emp_Id, Emp_email }
{ Emp_SSN, Emp_name, Emp_Id }

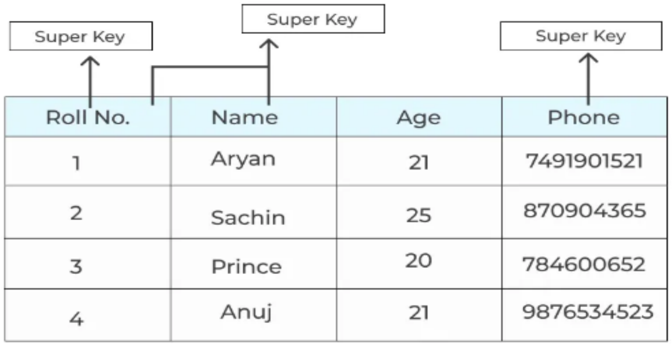
Example : Let's consider an STUDENT **table example where we have the following attribute**



Student

Roll_no	Name	Registration_no
1	Andrew	895
2	Angel	564
3	Augusto	567

1. {Roll_no}
2. {Registration_no}
3. {Roll_no, Registration_no},
4. {Roll_no, Name}
5. {Name, Registration_no}
6. {Roll_no, Name, Registration_no}


Super Key
Super Key
Super Key

Roll No.	Name	Age	Phone
1	Aryan	21	7491901521
2	Sachin	25	870904365
3	Prince	20	784600652
4	Anuj	21	9876534523

1. {Roll_no}
2. {Phone}
3. {Roll_no, Phone},
4. {Roll_no, Name}
5. {Roll no, Name, Age}
6. {Roll_no, Name, Age, Phone}
7. {Phone,Name}
8. {Phone,Age}
9. {Phone, Name, Age}

Example:
The given relation R(A, B, C, D, E, F) and check for super keys by following given dependencies:

Functional dependencies	**Super key**
AB->CDEF	**YES**
CD->ABEF	**YES**
CB->DF	**NO**
D->BC	**NO**

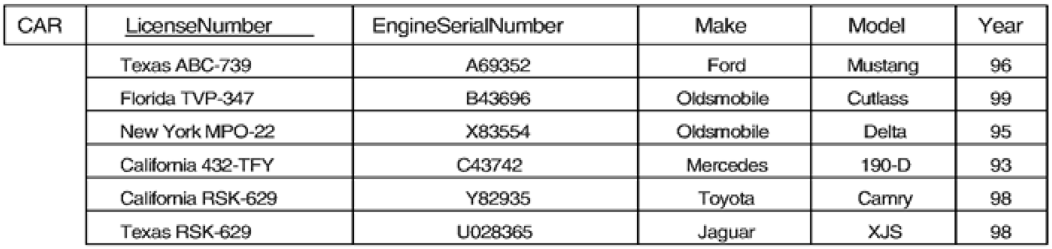
By Using key AB we can identify the rest of the attributes (CDEF) of the table.
Similarly, Key CD. But, by using key CB we can only identify D and F, not A and E.
Similarly key D.

2) ***Key of R:*** A "minimal" superkey; that is, a superkey K such that removal of any attribute from K results in a set of attributes that is not a superkey.

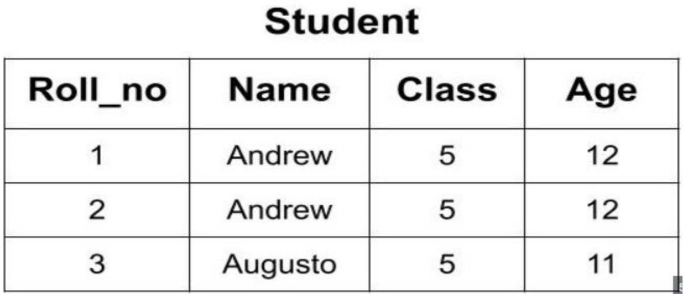
Example: The CAR relation schema:

CAR(State, Reg#, SerialNo, Make, Model, Year)
has two keys Key1 = {State, Reg#}, Key2 = {SerialNo}, which are also superkeys.
{SerialNo, Make} is a superkey but not a key.
If a relation has several candidate keys, one is chosen arbitrarily to be the primary key. The primary key attributes are underlined.

CAR relation with two candidate Keys LicenseNumber and EngineSerialNumber



CAR	LicenseNumber	EngineSerialNumber	Make	Model	Year
	Texas ABC-739	A69352	Ford	Mustang	96
	Florida TVP-347	B43696	Oldsmobile	Cutlass	99
	New York MPO-22	X83554	Oldsmobile	Delta	95
	California 432-TFY	C43742	Mercedes	190-D	93
	California RSK-629	Y82935	Toyota	Camry	98
	Texas RSK-629	U028365	Jaguar	XJS	98

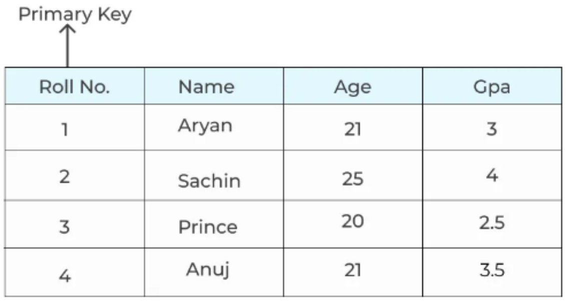


Student

Roll_no	Name	Class	Age
1	Andrew	5	12
2	Andrew	5	12
3	Augusto	5	11

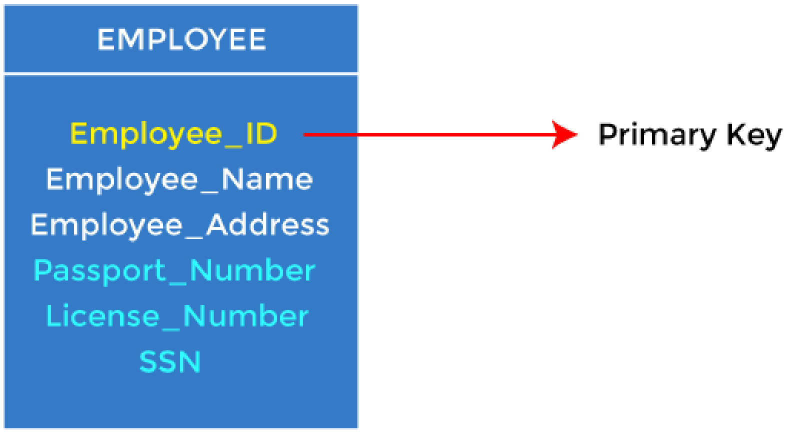
Primary Key

- It does not allow null values or blank.
- No duplicate value exists for that particular field.
- Two rows can't have the same primary key value
- A table cannot have more than one primary key.
- It is selected from a set of candidate keys.
- The minimal set of attributes is choose has primary key.
- ***Any candidate key can become a primary key.***
- It depends upon the requirements and is done by the Database Administrator (DBA).
- The value in a primary key column can never be modified or updated if any foreign key refers to that primary key.



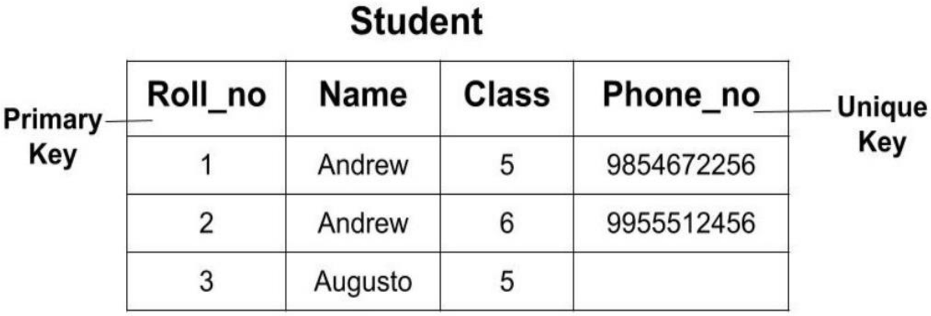
Primary Key

Roll No.	Name	Age	Gpa
1	Aryan	21	3
2	Sachin	25	4
3	Prince	20	2.5
4	Anuj	21	3.5



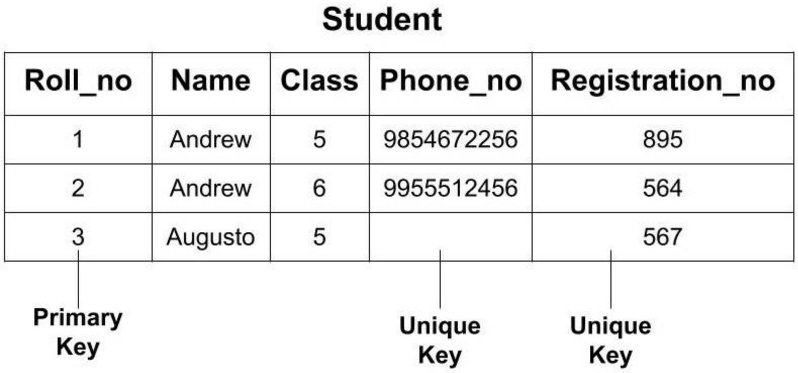
Unique Key

- It is similar to PK,
- It allows null values or blank.
- No duplicate value exists for that particular field.


Student

Roll_no	Name	Class	Phone_no
1	Andrew	5	9854672256
2	Andrew	6	9955512456
3	Augusto	5	

Primary Key
Unique Key


Student

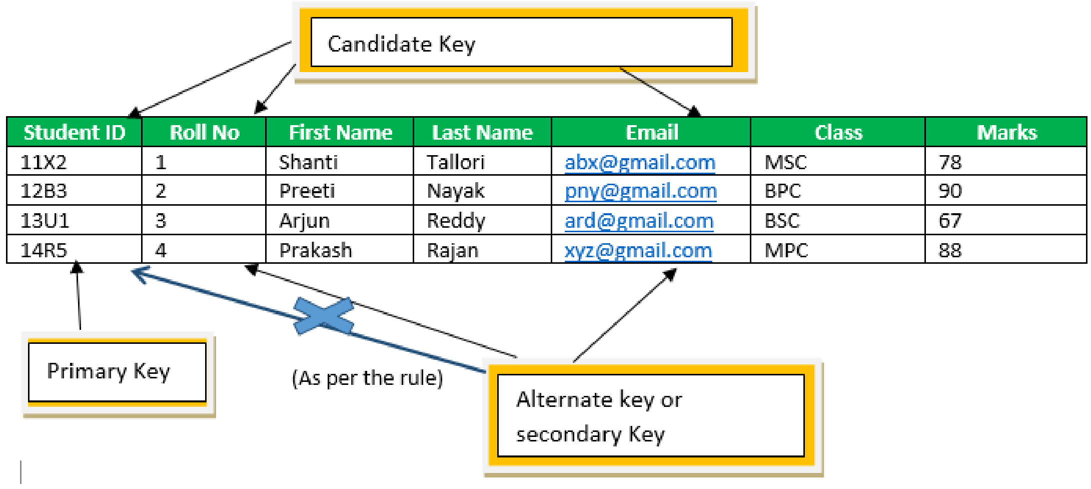
Roll_no	Name	Class	Phone_no	Registration_no
1	Andrew	5	9854672256	895
2	Andrew	6	9955512456	564
3	Augusto	5		567

Primary Key
Unique Key
Unique Key

Secondary Key /Alternate Key

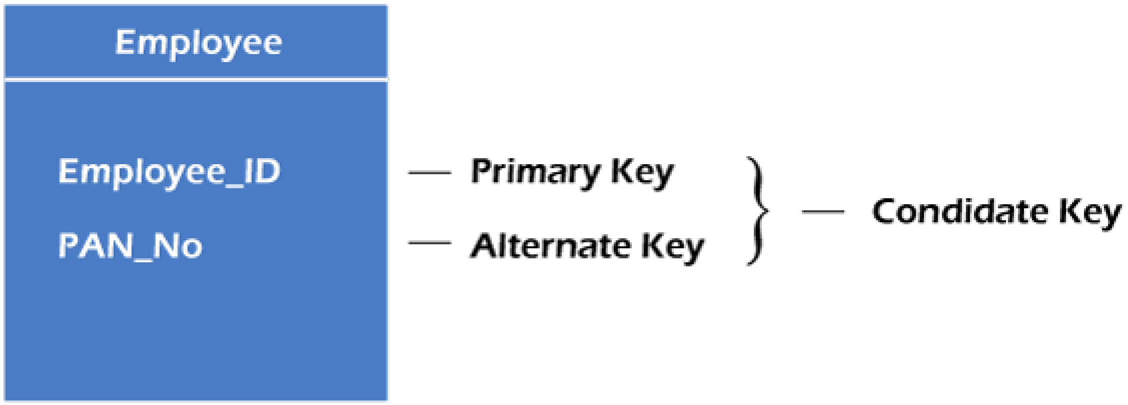
- An alternate key i.e. not used as PK. Or the candidate key other than the primary key is called an alternate key.
- All the keys which are not primary keys are called alternate keys / secondary key.
- These values are repeated.

- There may be one or more attributes or a combination of attributes that uniquely identify each tuple in a relation.

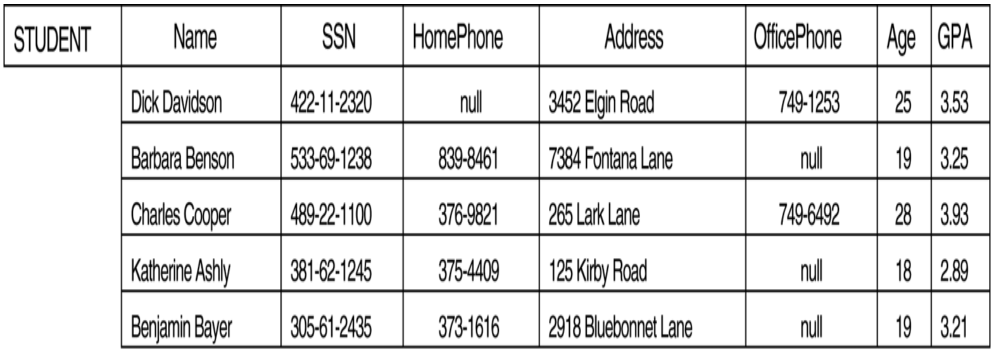


Student ID	Roll No	First Name	Last Name	Email	Class	Marks
11X2	1	Shanti	Tallori	abx@gmail.com	MSC	78
12B3	2	Preeti	Nayak	pny@gmail.com	BPC	90
13U1	3	Arjun	Reddy	ard@gmail.com	BSC	67
14R5	4	Prakash	Rajan	xyz@gmail.com	MPC	88

For example, Employee relation has two attributes, Employee_Id and PAN_No, that act as candidate keys. In this relation, Employee_Id is chosen as the primary key, so the other candidate key, PAN_No, acts as the Alternate key.



Example:- SNAME, and ADDRESS is Alternate keys



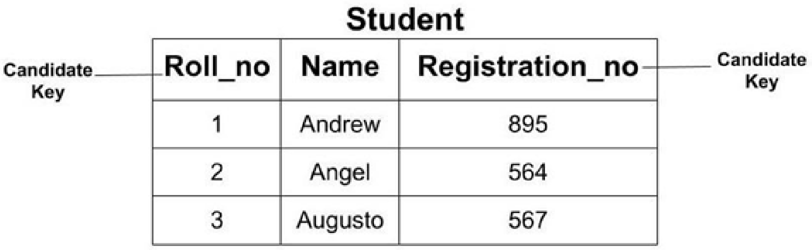
STUDENT	Name	SSN	HomePhone	Address	OfficePhone	Age	GPA
	Dick Davidson	422-11-2320	null	3452 Elgin Road	749-1253	25	3.53
	Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	null	19	3.25
	Charles Cooper	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
	Katherine Ashly	381-62-1245	375-4409	125 Kirby Road	null	18	2.89
	Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	null	19	3.21

- Consider the table shown above.
  STUD_NO, as well as PHONE both, are candidate keys for relation STUDENT but PHONE will be an alternate key
  (only one out of many candidate keys).

Candidate Key

- A relation schema may have more than one key; each is called a candidate key.
- A candidate key is an attribute or set of attributes that can uniquely identify a tuple.
- Except for the primary key, the remaining attributes are considered a candidate key. The candidate keys are as strong as the primary key.
- A candidate key which have no redundant attributes

Example :



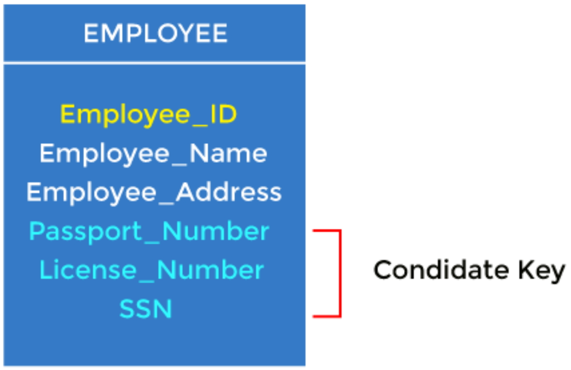
Student

Roll_no	Name	Registration_no
1	Andrew	895
2	Angel	564
3	Augusto	567

Candidate Key
Candidate Key

{Roll_no}: This key doesn't have any redundant or repeating attribute. So, it can be considered as a candidate key.
{Registration_no}: This key also doesn't have any repeating attribute. So, it can be considered as a candidate key.

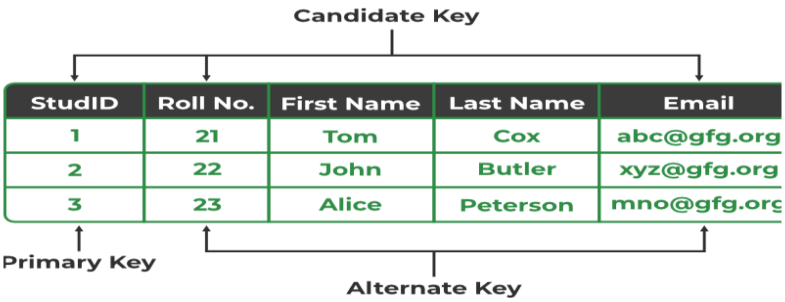
For example: In the EMPLOYEE table, id is best suited for the primary key. The rest of the attributes, like SSN, Passport_Number, License_Number, etc., are considered a candidate key.



EX: {SSN} is the only candidate key for EMPLOYEE, so it is also the PK.
Ex: CAR { LicenseNo, EngineNo, Make, Year, Color }. LicenseNo and EngineNo are The two Candidate key used to identify tuples in the relation.

Properties of Candidate key:

- It must contain unique values
- Candidate key in SQL may have multiple attributes
- Must not contain null values
- It should contain minimum fields to ensure uniqueness
- Uniquely identify each record in a table

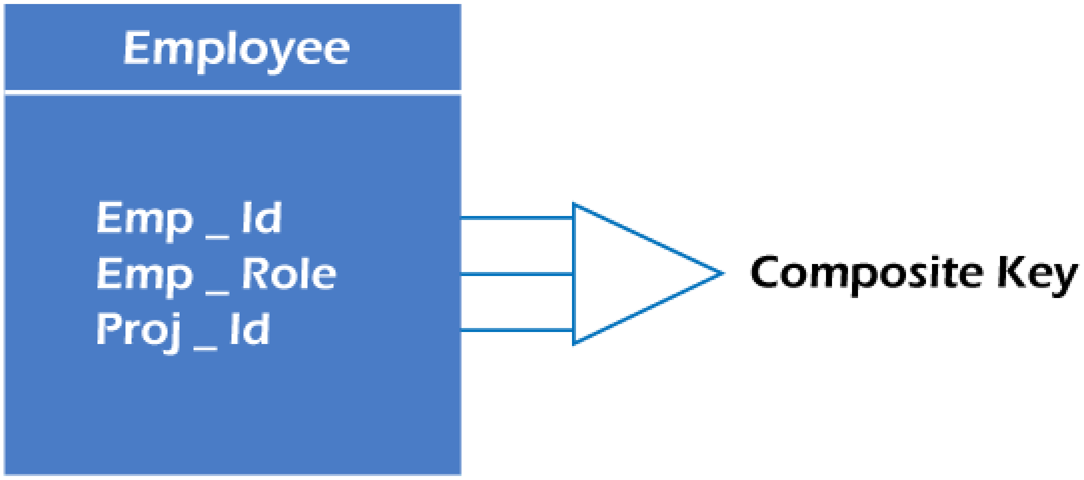


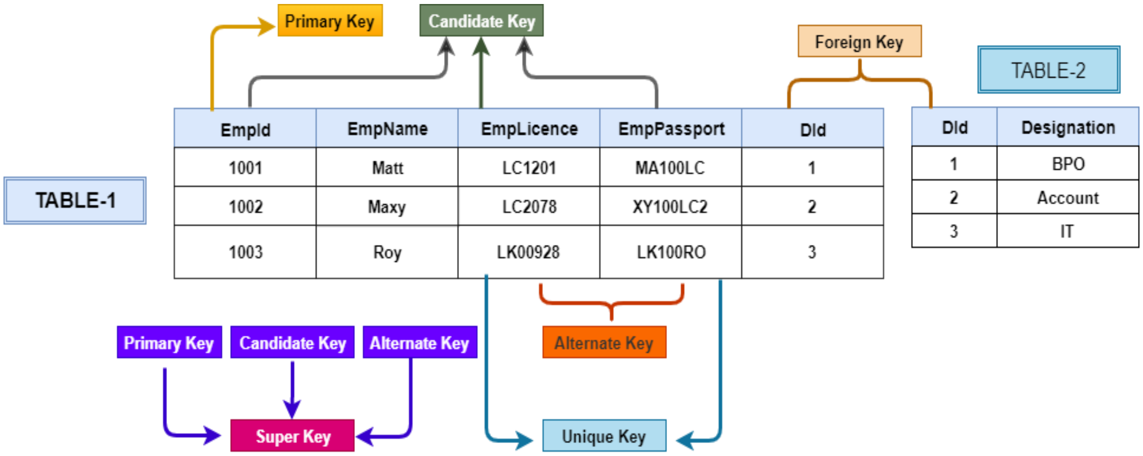
StudID	Roll No.	First Name	Last Name	Email
1	21	Tom	Cox	abc@gfg.org
2	22	John	Butler	xyz@gfg.org
3	23	Alice	Peterson	mno@gfg.org

Composite Key or Concatenated Key or Compound Key

- A combination of two or more fields that together comprise the primary key.
- Any one of the key attribute is composite attribute.
  Ex: CAR Relation Registration (Regno, State), VehicleId, Make, Model, Year, Color
  The two key attributes, Registration and VehicleId.

For example, in employee relations, we assume that an employee may be assigned multiple roles, and an employee may work on multiple projects simultaneously. So the primary key will be composed of all three attributes, namely Emp_ID, Emp_role, and Proj_ID in combination. So these attributes act as a composite key since the primary key comprises more than one attribute.



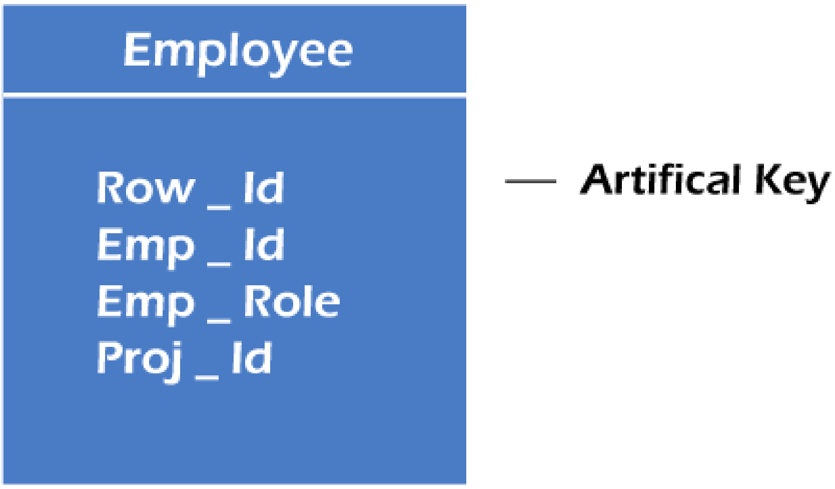


EmpId	EmpName	EmpLicence	EmpPassport	DId
1001	Matt	LC1201	MA100LC	1
1002	Maxy	LC2078	XY100LC2	2
1003	Roy	LK00928	LK100RO	3

DId	Designation
1	BPO
2	Account
3	IT

Artificial key/ Surrogate Keys

- The key created using arbitrarily assigned data are known as artificial keys.
- These keys are created when a primary key is large and complex and has no relationship with many other relations.
- The data values of the artificial keys are usually numbered in a serial order.
- An artificial key which aims to uniquely identify each record is called a surrogate key.
- This kind of partial key in dbms is unique because it is created when you don't have any natural primary key.
- They do not lend any meaning to the data in the table.
- Surrogate key in DBMS is usually an integer.
- A surrogate key is a value generated right before the record is inserted into a table
- A surrogate key is also known by various other names, which are *pseudo key*, *technical key*, *synthetic key*, *arbitrary unique identifier*, *entity identifier and database sequence number*.



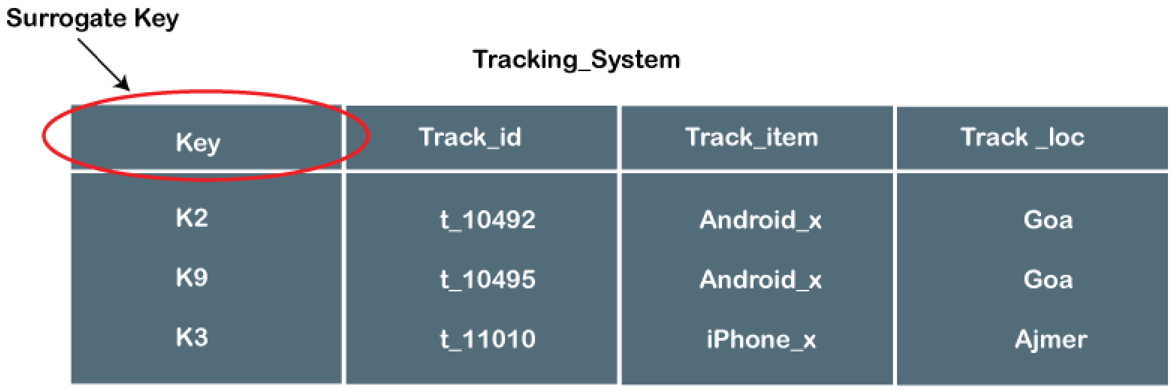
Surrogate keys in sql are allowed when

- No property has the parameter of the primary key.

- In the table when the primary key is too big or complicated.

Fname	Lastname	Start Time	End Time
Anne	Smith	09:00	18:00
Jack	Francis	08:00	17:00
Anna	McLean	11:00	20:00
Shown	Willam	14:00	23:00

Above, given example, shown shift timings of the different employee. In this example, a surrogate key is needed to uniquely identify each employee.


Surrogate Key

Tracking_System

Key	Track_id	Track_item	Track _loc
K2	t_10492	Android_x	Goa
K9	t_10495	Android_x	Goa
K3	t_11010	iPhone_x	Ajmer

Entity Integrity Constraint

- **Relational Database Schema**:

A set S of relation schemas that belong to the same database. S is the name of the database.
S = {R1, R2, ..., Rn}

- **Entity Integrity:**

The primary key attributes PK of each relation schema R in S cannot have null values in any tuple of r(R).
This is because primary key values are used to identify the individual tuples.
t[PK] ≠ null for any tuple t in r(R)

- **Note:** **Other attributes of R may be similarly constrained to disallow null values, even though they are not members of the primary key.**
- **These constraints involve a single relation**.

Referential Integrity Constraint

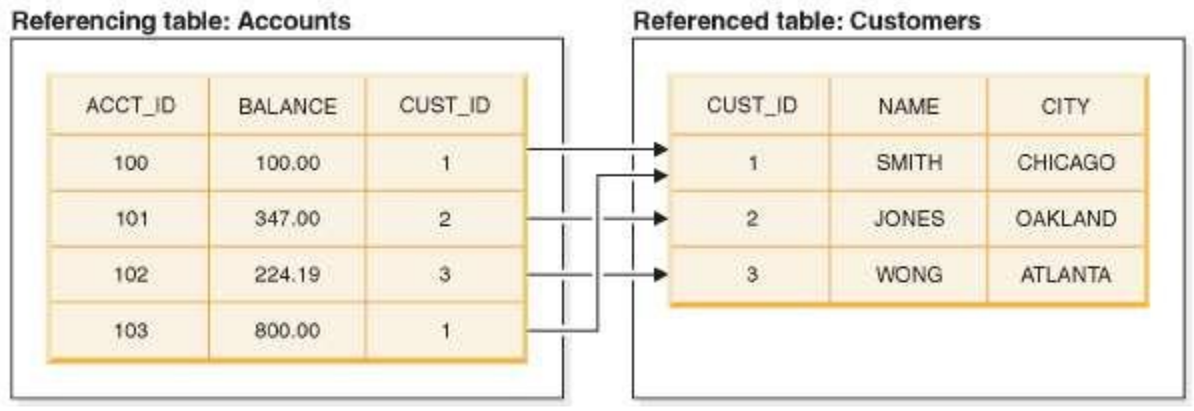
- A constraint is specified between (or involving) two relations.
- It used to maintain the consistency among tuples in the two relations.
- The referential integrity constraint states that a tuple in one relation that refers to another relation must refer to an existing tuple in the relation.

Foreign Key (Referencing Key)

- FOREIGN KEY is a column that creates a relationship between two tables or within the table.
- A field in a table that refers to a PK key field in another table. Or it can be defined as a copy of a primary key in another table.
- It can have null value and duplication is allowed.
- Used to specify a relationship among tuples in two relations: the referencing relation and the referenced relation.

- Tuples in the referencing relation R1 have attributes FK (called foreign key attributes) that reference the primary key attributes PK of the referenced relation R2. A tuple t1 in R1 is said to reference a tuple t2 in R2 if t1[FK] = t2[PK].

- **The value in the foreign key column (or columns) FK of the referencing relation R1 can be either:**
  (1) a value of an existing primary key value of the corresponding primary key PK in the referenced relation R2, or..
  (2) a null.
  In case (2), the FK in R1 should not be a part of its own primary key.
  A referential integrity constraint can be displayed in a relational database schema as a directed arc from R1.FK to R2.

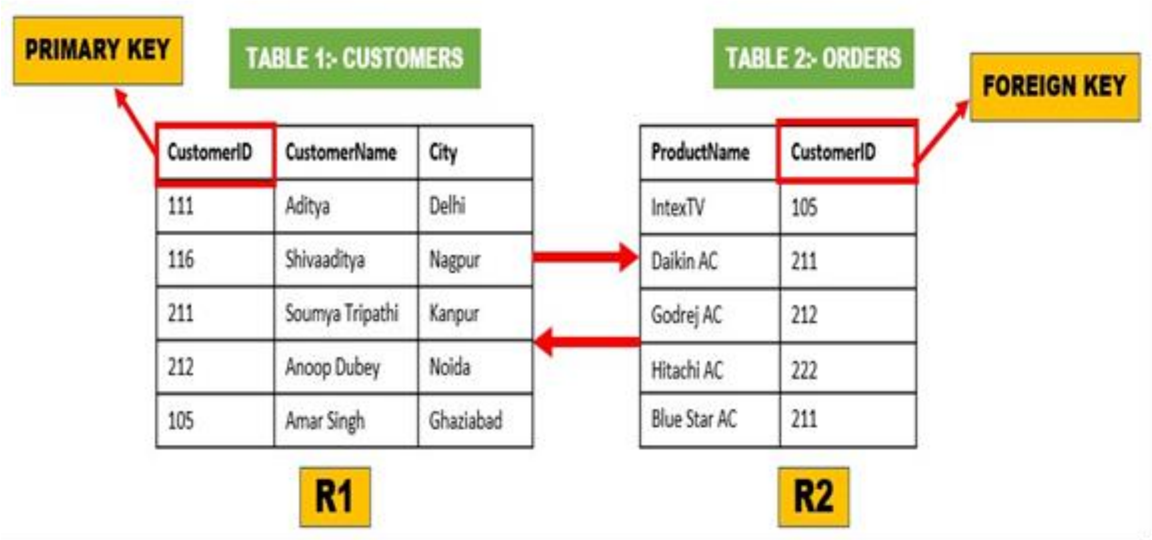


Referencing table: Accounts

ACCT_ID	BALANCE	CUST_ID
100	100.00	1
101	347.00	2
102	224.19	3
103	800.00	1

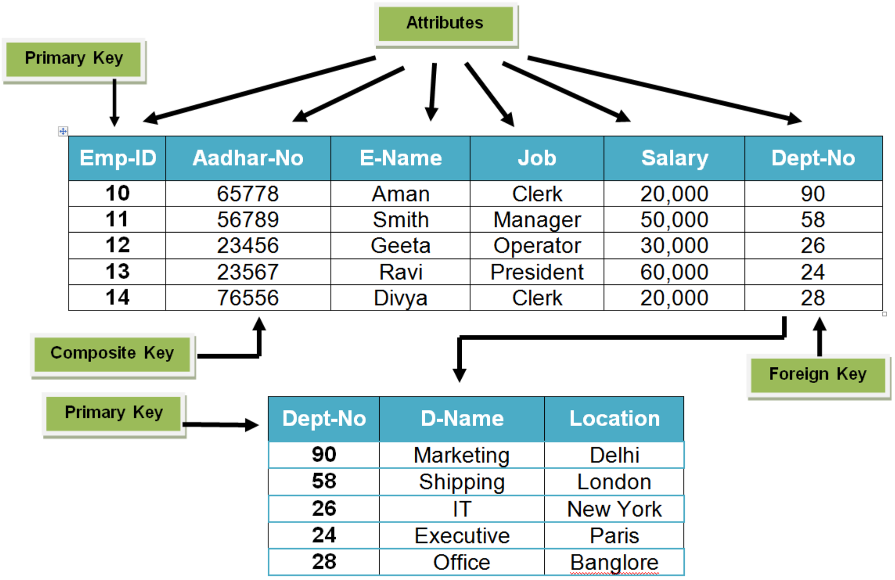
Referenced table: Customers

CUST_ID	NAME	CITY
1	SMITH	CHICAGO
2	JONES	OAKLAND
3	WONG	ATLANTA



CustomerID	CustomerName	City
111	Aditya	Delhi
116	Shivaaditya	Nagpur
211	Soumya Tripathi	Kanpur
212	Anoop Dubey	Noida
105	Amar Singh	Ghaziabad

ProductName	CustomerID
IntexTV	105
Daikin AC	211
Godrej AC	212
Hitachi AC	222
Blue Star AC	211



Emp-ID	Aadhar-No	E-Name	Job	Salary	Dept-No
10	65778	Aman	Clerk	20,000	90
11	56789	Smith	Manager	50,000	58
12	23456	Geeta	Operator	30,000	26
13	23567	Ravi	President	60,000	24
14	76556	Divya	Clerk	20,000	28

Dept-No	D-Name	Location
90	Marketing	Delhi
58	Shipping	London
26	IT	New York
24	Executive	Paris
28	Office	Banglore

Other Types of Constraints

Semantic Integrity Constraints:

- based on application semantics and cannot be expressed by the model per se
- E.g., “the max. no. of hours per employee for all projects he or she works on is 56 hrs per week”
- A constraint specification language may have to be used to express these
- SQL-99 allows triggers and ASSERTIONS to allow for some of these.

Functional Dependency Constraint:

- A functional relationship among two sets of attributes X and Y.

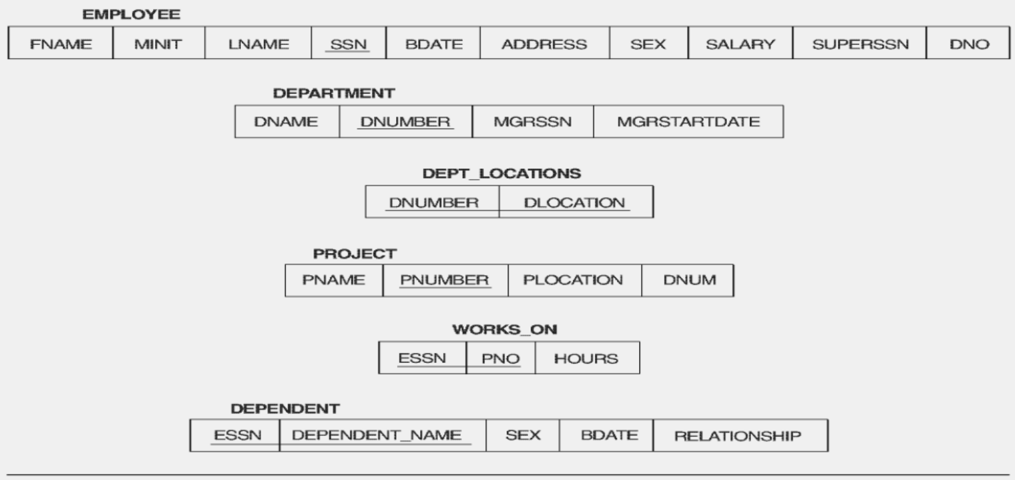
- This constraint specifies that the value of X determines the value of Y in all states of a relation.
- It is denoted by X →Y.

Relational database schema & Integrity constraints

A relational DB schema S is a set of relation schema S={R1,R2,….Rm} & set of integrity constraints(IC).

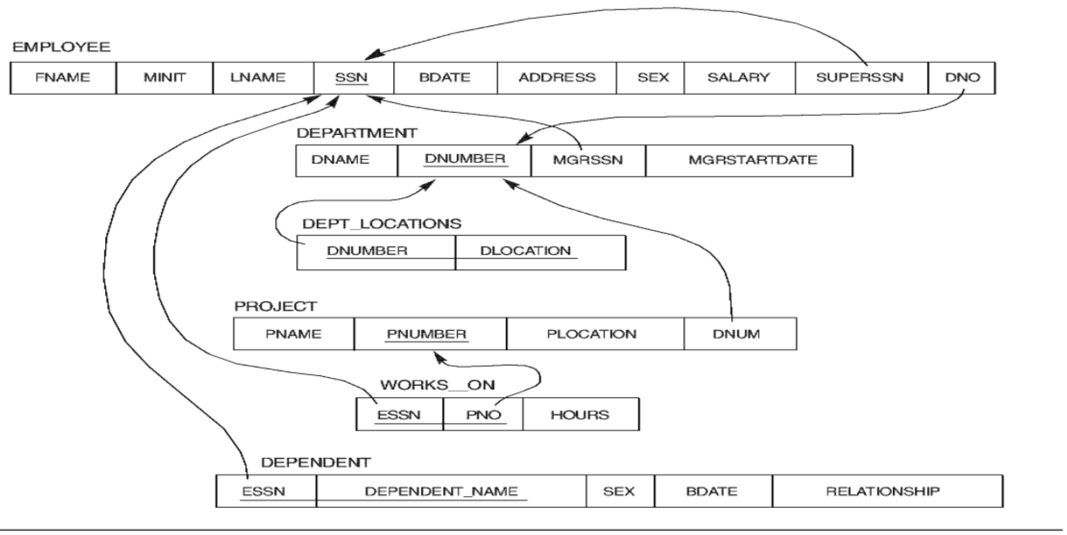
A relational DB instance DB={r1,r2,…,rm} such that the r_i is an instance of R_i & r_i relation satisfy the integrity constraints specified in IC.

Figure 7.5 Schema diagram for the COMPANY relational database schema; the primary keys are underlined.



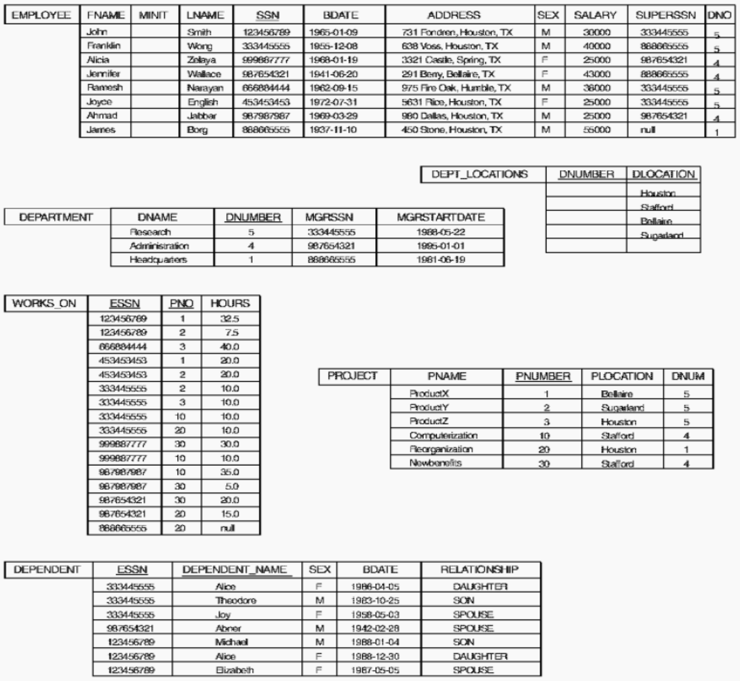
© Addison Wesley Longman, Inc. 2000, Elmasri/Navathe, Fundamentals of Database Systems, Third Edition

Figure 7.7 Referential integrity constraints displayed on the COMPANY relational database schema diagram.



© Addison Wesley Longman, Inc. 2000, Elmasri/Navathe, Fundamentals of Database Systems, Third Edition

Figure 7.6 One possible relational database state corresponding to the COMPANY schema.



EMPLOYEE	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
	John		Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
	Franklin		Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
	Alicia		Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
	Jennifer		Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
	Ramesh		Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
	Joyce		English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
	Ahmad		Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
	James		Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	null	1

DEPT_LOCATIONS	DNUMBER	DLOCATION
		Houston
		Stafford
		Bellaire
		Sugarland

DEPARTMENT	DNAME	DNUMBER	MGRSSN	MGRSTARTDATE
	Research	5	333445555	1988-05-22
	Administration	4	987654321	1995-01-01
	Headquarters	1	888665555	1981-06-19

WORKS_ON	ESSN	PNO	HOURS
	123456789	1	32.5
	123456789	2	7.5
	666884444	3	40.0
	453453453	1	20.0
	453453453	2	20.0
	333445555	2	10.0
	333445555	3	10.0
	333445555	10	10.0
	333445555	20	10.0
	999887777	30	30.0
	999887777	10	10.0
	987987987	10	35.0
	987987987	30	5.0
	987654321	30	20.0
	987654321	20	15.0
	888665555	20	null

PROJECT	PNAME	PNUMBER	PLOCATION	DNUM
	ProductX	1	Bellaire	5
	ProductY	2	Sugarland	5
	ProductZ	3	Houston	5
	Computerization	10	Stafford	4
	Reorganization	20	Houston	1
	Newbenefits	30	Stafford	4

DEPENDENT	ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP
	333445555	Alice	F	1986-04-05	DAUGHTER
	333445555	Theodore	M	1983-10-25	SON
	333445555	Joy	F	1958-05-03	SPOUSE
	987654321	Abner	M	1942-02-28	SPOUSE
	123456789	Michael	M	1988-01-04	SON
	123456789	Alice	F	1988-12-30	DAUGHTER
	123456789	Elizabeth	F	1967-05-05	SPOUSE

© Addison Wesley Longman, Inc. 2000, Elmasri/Navathe, Fundamentals of Database Systems, Third Edition

Query is a question or requesting information. Query language is a language which is used to retrieve information from a database.

Query language is divided into two types −

- Procedural language
- Non-procedural language

Procedural language

Information is retrieved from the database by specifying the sequence of operations to be performed.

For Example: Relational algebra.

Structure Query language (SQL) is based on **relational algebra**.

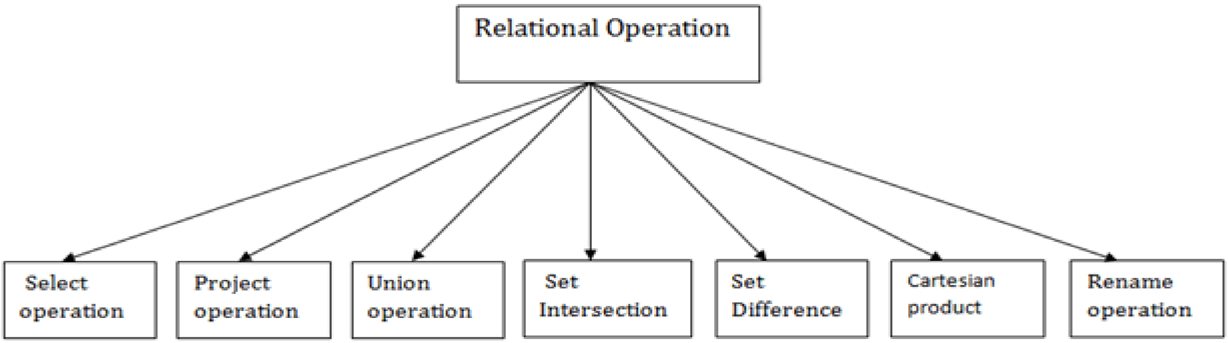
Relational algebra consists of a set of operations that take one or two relations as an input and produces a new relation as output.

Relational Algebra

Relational algebra is a procedural query language, which takes instances of relations as input and yields instances of relations as output. It uses operators to perform queries. An operator can be either unary or binary. They accept relations as their input and yield relations as their output. Relational algebra is performed recursively on a relation and intermediate results are also considered relations.

The different types of relational algebra operations are as follows −

- Select operation
- Project operation
- Rename operation
- Union operation
- Intersection operation
- Difference operation
- Cartesian product operation
- Join operation
- Division operation



Operation(Symbols)	Purpose
Select(σ)	The SELECT operation is used for selecting a subset of the tuples according to a given selection condition
Projection(π)	The projection eliminates all attributes of the input relation but those mentioned in the projection list.
Union Operation(∪)	UNION is symbolized by symbol. It includes all tuples that are in tables A or in B.
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.
Intersection(∩)	Intersection defines a relation consisting of a set of all tuple that are in both A and B.
Cartesian Product(X)	Cartesian operation is helpful to merge columns from two relations.

Operation(Symbols)	Purpose
Inner Join	Inner join, includes only those tuples that satisfy the matching criteria.
Theta Join(θ)	The general case of JOIN operation is called a Theta join. It is denoted by symbol θ.
EQUI Join	When a theta join uses only equivalence condition, it becomes a equi join.
Natural Join(⋈)	Natural join can only be performed if there is a common attribute (column) between the relations.
Outer Join	In an outer join, along with tuples that satisfy the matching criteria.
Left Outer Join(⟕)	In the left outer join, operation allows keeping all tuple in the left relation.
Right Outer join(⟖)	In the right outer join, operation allows keeping all tuple in the right relation.
Full Outer Join(⟗)	In a full outer join, all tuples from both relations are included in the result irrespective of the matching condition.

R1

sid	bid	day
22	101	10/10/99
58	103	11/12/99

S1

sid	sname	rating	age
22	Anu	7	45.0
31	Laxmi	8	55.5
58	Roopa	10	35.0

S2

sid	sname	rating	age
28	Yamuna	9	35.0
31	Laxmi	8	55.0
44	Geeta	5	35.0
58	Roopa	10	35.0

SELECT (σ):

Unary operator (one relation as operand)

It **selects tuples** that satisfy the given predicate from a relation.

p is prepositional logic formula which may use connectors like and, or, and not. These terms may use relational operators like − =, ≠, ≥, < , >, ≤.

$$\sigma_p(r)$$

σ is the predicate

r stands for relation which is the name of the table

p is prepositional logic

OR

σ<selection condition> R

where <selection condition>

• may have Boolean conditions AND, OR, and NOT

• has clauses of the form:

<attribute name> <comparison op> <constant value>

or

<attribute name> <comparison op> <attribute name>

1. Selects tuples from S1 where sname = ‘Anu’.

$\boldsymbol{\sigma}_{\textbf{Sname = "Anu"}}$ **(S1)**

2. $\boldsymbol{\sigma}_{\textit{subject = "database"}}$**(Books)**

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

3. $\boldsymbol{\sigma}_{\textbf{subject = "database" and price = "450"}}$**(Books)**

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

4. $\sigma_{\text{subject = "database" and price = "450" or year > "2010"}}$(Books)

Output − Selects tuples from books where subject is 'database' and 'price' is 450 or those books published after 2010.

Projection(π) :

It projects column(s) that satisfy a given predicate.

$\prod_{\mathbf{A1,\ A2,\ An}} \mathbf{(r)}$

Where A_1, A_2 , A_n are attribute names of relation **r**.

Uary operator (one relation as operand)

Keeps specified attributes and discards the others:

π*<attribute list> R*

□ Duplicate elimination

• Result of PROJECT operation is a set of distinct tuples

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

sname	rating
Yamuna	9
Laxmi	8
Geeta	5
Roopa	10

$\pi_{\mathbf{sname,\ rating}}\boldsymbol{(S2)}$

sname	rating
Yamuna	9
Roopa	10

$\pi_{sname,\ rating}(S2)\ (\sigma_{rating\ >\ 8}(S2))$

Rename(ρ):

Rename is a unary operation used for renaming attributes of a relation.

ρ(a/b)R will rename the attribute 'b' of the relation by 'a'.
Or

ρ_{x} **(E)**

Where the result of expression **E** is saved with name of **x**.

Renaming can be used by three methods, which are as follows −

- Changing name of the relation.
- Changing name of the attribute.
- Changing both.

Unary RENAME operator

• Rename relation

$\rho S\ (\ R)$

• Rename attributes

$\rho(B1,B2,\ldots Bn)\ (R)$

• Rename relation and its attributes

$\rho S(B1,B2,\ldots,Bn)\ (R)$

Union, Intersection, Set Difference

It performs binary union between two given relations and is defined as

Union : Builds a relation from tuples appearing in either or both of the specified relations.

$r \cup s = \{ t \mid t \in r \text{ or } t \in s\}$

Notation − $r \cup s$

Where **r** and **s** are either database relations or relation result set (temporary relation).

NOTE:

For a union operation to be valid, the following conditions must hold −

- **r**, and **s** must have the same number of attributes.
- Attribute domains must be compatible.
- Duplicate tuples are automatically eliminated.

sid	sname	rating	age
22	Anu	7	45.0
31	Laxmi	8	55.0
44	Geeta	5	35.0
58	Roopa	10	35.0
28	Yamuna	9	35.0

$S1 \cup S2$

Projects the names of the authors who have either written a book or an article or both.

$\prod_{\text{author}}$ (Books) $\cup$ $\prod_{\text{author}}$ (Articles)

Intersection: Builds a relation consisting of tuples that appear in both the relations.

sid	sname	rating	age
31	Laxmi	8	55.0
58	Roopa	10	35.0

$S1 \cap S2$

Provides the name of authors who have written books and articles both.

$\prod_{\text{author}}$ (Books) $\bigcap$ $\prod_{\text{author}}$ (Articles)

Set Difference: Builds a relation of tuples appearing in the first but not the second of two specified relations.

S1 – S2

sid	sname	rating	age
22	Anu	7	45.0

Provides the name of authors who have written books not articles.

$\prod_{\text{author}} (\text{Books}) - \prod_{\text{author}} (\text{Articles})$

Cross- Product / Cartesian Product (X)

Cross-Product:
Builds a relation from two specified relations, such that it consists of all possible relations, one from each of the two relations.

Combines information of two different relations into one.

Notation − r X s

Where **r** and **s** are relations and their output will be defined as −

r X s = { q t | q ∈ r and t ∈ s}

Each row of S1 is paired with each row of R1.

Result schema has one field per field of S1 and R1, with field names `inherited' if possible.
– *Conflict* : Both S1 and R1 have a field called *sid* .

sid	sname	rating	age	sid	bid	day
22	Anu	7	45.0	22	101	10/10/99
22	Anu	7	45.0	58	103	11/12/99
31	Laxmi	8	55.5	22	101	10/10/99
31	Laxmi	8	55.5	58	103	11/12/99
58	Roopa	10	35.0	22	101	10/10/99
58	Roopa	10	35.0	58	103	11/12/99

A relation, which shows all the books and articles written by tutorialspoint.

Division Operator (÷):

The division operator is used for queries which involve the 'all'.

R1 ÷ R2 = tuples of R1 associated with all tuples of R2.

OR

Division operator A÷B or A/B can be applied if and only if:

- Attributes of B is proper subset of Attributes of A.
- The relation returned by division operator will have attributes = (All attributes of A – All Attributes of B)
- The relation returned by division operator will return those tuples from relation A which are associated to every B's tuple.

A

x	**y**
a	1
b	2
a	2
d	4

÷∏

B

y
1
2

The resultant of A/B is

A ÷ B

x
a

Division can be expressed in terms of **Cross Product** , **Set Difference** and **Projection.**
In the above example , for A/B , compute all x values that are not disqualified by some y in B.

x value is disqualified if attaching y value from B, we obtain xy tuple that is not in A.

Disqualified x values: $\prod_x((\prod_x(A) \times B) - A)$
So A/B = $\prod_x(A)$ − all disqualified tuples
A/B = $\prod_x(A) - \prod x((\prod x(A) \times B) - A)$
In the above example , disqualified tuples are

b	2
d	4

So, the resultant is

x
a

Example2:
Retrieve the name of the subject that is taught in all courses.

Table1

Name	**Course**
System	Btech
Database	Mtech
Database	Btech
Algebra	Btech

Table2

Course
Btech
Mtech
Btech

Name
database

The resulting operation must have all combinations of tuples of relation S that are present in the first relation or R.
Example3
Retrieve names of employees who work on all the projects that John Smith works on.

Consider the Employee table given below −

Name	Eno	Pno
John	123	P1
Smith	123	P2
A	121	P3

÷

Works on the following −

Eno	Pno	Pname
123	P1	Market
123	P2	Sales

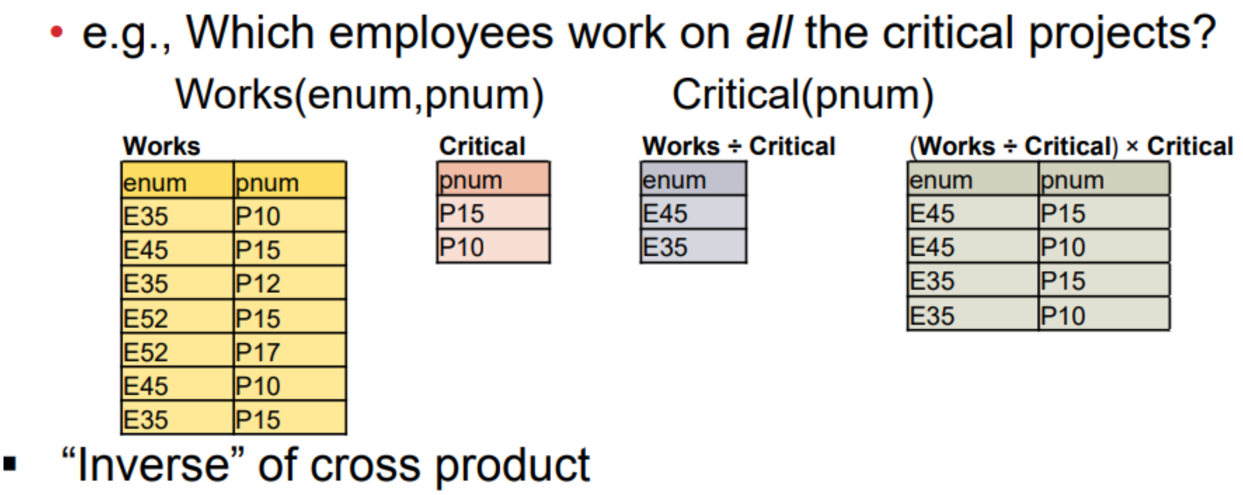
=

The result is as follows

Eno
123

The expression is as follows

Smith <- ΠPno(σEname = 'john smith' (employee * works on Pno=Eno))



- e.g., Which employees work on *all* the critical projects?

Works(enum,pnum) Critical(pnum)

Works

enum	pnum
E35	P10
E45	P15
E35	P12
E52	P15
E52	P17
E45	P10
E35	P15

Critical

pnum
P15
P10

Works ÷ Critical

enum
E45
E35

(Works ÷ Critical) × Critical

enum	pnum
E45	P15
E45	P10
E35	P15
E35	P10

- "Inverse" of cross product

Join Operations:

A Join operation combines related tuples from different relations, if and only if a given join condition is satisfied. It is denoted by ⋈.

Example:

EMPLOYEE

EMP_CODE	EMP_NAME
101	Stephan
102	Jack
103	Harry

SALARY

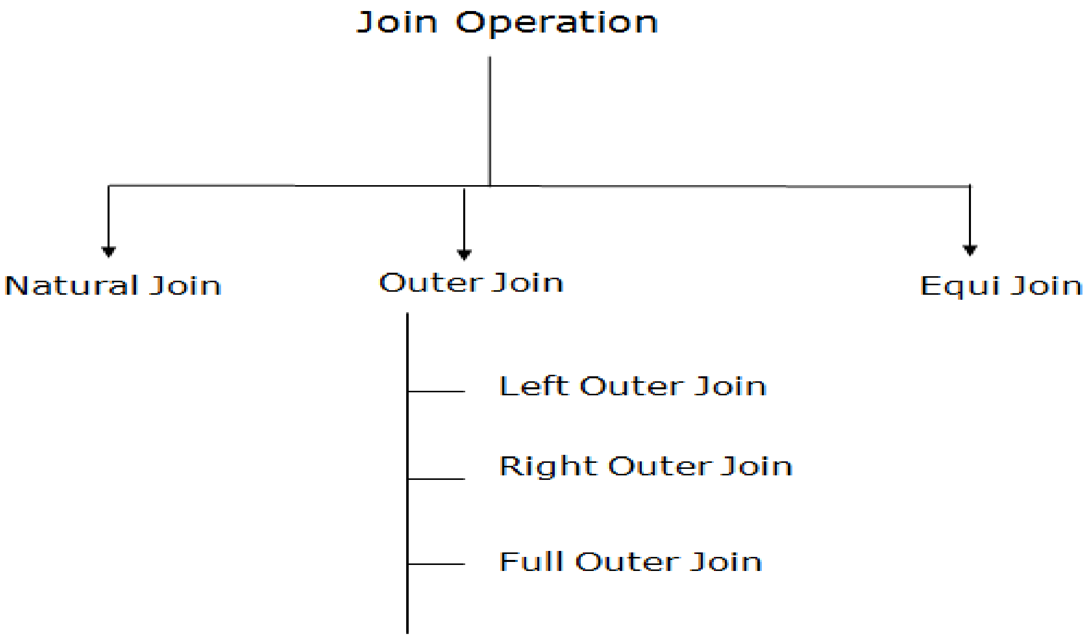
EMP_CODE	SALARY
101	50000
102	30000
103	25000

1. Operation: (EMPLOYEE ⋈ SALARY)

2.

EMP_CODE	EMP_NAME	SALARY
101	Stephan	50000
102	Jack	30000
103	Harry	25000

Types of Join operations:



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

1. Natural Join:

- A natural join is the set of tuples of all combinations in R and S that are equal on their common attribute names.
- or
- 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.

- It is denoted by ⋈.

Example: Let's use the above EMPLOYEE table and SALARY table:

Input:

1. ∏EMP_NAME, SALARY (EMPLOYEE ⋈ SALARY)

Output:

EMP_NAME	SALARY
Stephan	50000
Jack	30000
Harry	25000

Example2:

EMP

Name	ID	Dept_Name
A	120	IT
B	125	HR
C	110	Sales
D	111	IT

DEPT

Dept_Name	Manager
Sales	Y
Production	Z
IT	A

Natural join between EMP and DEPT with condition :

EMP.Dept_Name = DEPT.Dept_Name

EMP ⋈ DEPT

Name	ID	Dept_Name	Manager
A	120	IT	A
C	110	Sales	Y
D	111	IT	A

2.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:

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

2. Conditional Join: Conditional join works similarly to natural join. In natural join, by default condition is equal between common attributes while in conditional join we can specify any condition such as greater than, less than, or not equal.
Example:

R

ID	Sex	Marks
1	F	45
2	F	55
3	F	60

S

ID	Sex	Marks
10	M	20
11	M	22
12	M	59

Join between R and S with condition **R.marks >= S.marks**

R.ID	R.Sex	R.Marks	S.ID	S.Sex	S.Marks
1	F	45	10	M	20
1	F	45	11	M	22
2	F	55	10	M	20
2	F	55	11	M	22
3	F	60	10	M	20
3	F	60	11	M	22
3	F	60	12	M	59

3. Outer Join:

The outer join operation is an extension of the join operation. It is used to deal with missing information.

Example:

EMPLOYEE

EMP_NAME	STREET	CITY

Ram	Civil line	Mumbai
Shyam	Park street	Kolkata
Ravi	M.G. Street	Delhi
Hari	Nehru nagar	Hyderabad

FACT_WORKERS

EMP_NAME	BRANCH	SALARY
Ram	Infosys	10000
Shyam	Wipro	20000
Kuber	HCL	30000
Hari	TCS	50000

Input:

1. (EMPLOYEE ⋈ FACT_WORKERS)

Output:

EMP_NAME	STREET	CITY	BRANCH	SALARY
Ram	Civil line	Mumbai	Infosys	10000
Shyam	Park street	Kolkata	Wipro	20000
Hari	Nehru nagar	Hyderabad	TCS	50000

An outer join is basically of three types:

a. **Left outer join**
b. **Right outer join**
c. **Full outer join**

a. Left outer join:

- Left outer join contains the set of tuples of all combinations in R and S that are equal on their common attribute names.
- In the left outer join, tuples in R have no matching tuples in S.
- It is denoted by ⟕.

Example: Using the above EMPLOYEE table and FACT_WORKERS table

Input:

1. EMPLOYEE ⟕ FACT_WORKERS

EMP_NAME	STREET	CITY	BRANCH	SALARY
Ram	Civil line	Mumbai	Infosys	10000
Shyam	Park street	Kolkata	Wipro	20000
Hari	Nehru street	Hyderabad	TCS	50000
Ravi	M.G. Street	Delhi	NULL	NULL

b. Right outer join:

- Right outer join contains the set of tuples of all combinations in R and S that are equal on their common attribute names.
- In right outer join, tuples in S have no matching tuples in R.
- It is denoted by ⟖.

Example: Using the above EMPLOYEE table and FACT_WORKERS Relation

Input:

1. EMPLOYEE ⟖ FACT_WORKERS

Output:

EMP_NAME	BRANCH	SALARY	STREET	CITY
Ram	Infosys	10000	Civil line	Mumbai
Shyam	Wipro	20000	Park street	Kolkata
Hari	TCS	50000	Nehru street	Hyderabad
Kuber	HCL	30000	NULL	NULL

c. Full outer join:

- Full outer join is like a left or right join except that it contains all rows from both tables.
- In full outer join, tuples in R that have no matching tuples in S and tuples in S that have no matching tuples in R in their common attribute name.
- It is denoted by ⟗.

Example: Using the above EMPLOYEE table and FACT_WORKERS table

Input:

1. EMPLOYEE ⟗ FACT_WORKERS

Output:

EMP_NAME	STREET	CITY	BRANCH	SALARY
Ram	Civil line	Mumbai	Infosys	10000
Shyam	Park street	Kolkata	Wipro	20000
Hari	Nehru street	Hyderabad	TCS	50000
Ravi	M.G. Street	Delhi	NULL	NULL
Kuber	NULL	NULL	HCL	30000

4. Equi join:

It is also known as an inner join. It is the most common join. It is based on matched data as per the equality condition. The equi join uses the comparison **operator(=).**

Example:

CUSTOMER RELATION

CLASS_ID	NAME
1	John
2	Harry
3	Jackson

PRODUCT

PRODUCT_ID	CITY
1	Delhi
2	Mumbai
3	Noida

Input:

1. **CUSTOMER ⋈ CUSTOMER.CLASS_ID= PRODUCT.PRODUCT_ID PRODUCT**
2. **Output:**

CLASS_ID	NAME	PRODUCT_ID	CITY
1	John	1	Delhi
2	Harry	2	Mumbai

3	Harry	3	Noida

5. Inner Join

An INNER JOIN is the default join which retrieves the intersection of two tables. It compares each row of the first table with each row of the second table. If the pairs of these rows satisfy the join-predicate, they are joined together.

- Other Joins

In addition to these there are two more joins −

- SELF JOIN − is used to join a table to itself as if the table were two tables, temporarily renaming at least one table in the SQL statement.
- CROSS Join − returns the Cartesian product of the sets of records from the two or more joined tables.

Examples of Queries in Relational Algebra

Query 1. Retrieve the name and address of all employees who work for the ‘Research’ department.

$\text{RESEARCH\_DEPT} \leftarrow \sigma_{\text{Dname='Research'}}(\text{DEPARTMENT})$
$\text{RESEARCH\_EMPS} \leftarrow (\text{RESEARCH\_DEPT} \bowtie_{\text{Dnumber=Dno}} \text{EMPLOYEE})$
$\text{RESULT} \leftarrow \pi_{\text{Fname, Lname, Address}}(\text{RESEARCH\_EMPS})$

As a single in-line expression, this query becomes:

$^{\pi}\text{Fname, Lname, Address}\ ^{(\sigma}\text{Dname='Research'}^{\text{(DEPARTMENT}}\ \bowtie$
$\text{Dnumber=Dno}^{\text{(EMPLOYEE))}}$

This query could be specified in other ways; for example, the order of the JOIN and SELECT operations could be reversed, or the JOIN could be replaced by a NATURAL JOIN after renaming one of the join attributes to match the other join attribute name.

Query 2. For every project located in ‘Stafford’, list the project number, the controlling department number, and the department manager’s last name, address, and birth date.

$\text{STAFFORD\_PROJS} \leftarrow \sigma_{\text{Plocation='Stafford'}}(\text{PROJECT})$

$\text{CONTR\_DEPTS} \leftarrow (\text{STAFFORD\_PROJS} \bowtie_{\text{Dnum=Dnumber}} \text{DEPARTMENT})$

$\text{PROJ\_DEPT\_MGRS} \leftarrow (\text{CONTR\_DEPTS} \bowtie_{\text{Mgr\_ssn=Ssn}} \text{EMPLOYEE})$

$\text{RESULT} \leftarrow \pi\, \text{Pnumber, Dnum, Lname, Address, Bdate}^{(\text{PROJ\_DEPT\_MGRS})}$

In this example, we first select the projects located in Stafford, then join them with their controlling departments, and then join the result with the department man-agers. Finally, we apply a project operation on the desired attributes.

Query 3. Find the names of employees who work on *all* the projects controlled by department number 5.

$\text{DEPT5\_PROJS} \leftarrow \rho_{(\text{Pno})}(\pi_{\text{Pnumber}}(\sigma_{\text{Dnum=5}}(\text{PROJECT})))$

$\text{EMP\_PROJ} \leftarrow \rho_{(\text{Ssn, Pno})}(\pi_{\text{Essn, Pno}}(\text{WORKS\_ON}))$

$\text{RESULT\_EMP\_SSNS} \leftarrow \text{EMP\_PROJ} \div \text{DEPT5\_PROJS}$

$\text{RESULT} \leftarrow \pi_{\text{Lname, Fname}}(\text{RESULT\_EMP\_SSNS} * \text{EMPLOYEE})$

In this query, we first create a table DEPT5_PROJS that contains the project numbers of all projects controlled by department 5. Then we create a table EMP_PROJ that holds (Ssn, Pno) tuples, and apply the division operation. Notice that we renamed the attributes so that they will be correctly used in the division operation. Finally, we join the result of the division, which holds only Ssn values, with the EMPLOYEE table to retrieve the desired attributes from EMPLOYEE.

Query 4. Make a list of project numbers for projects that involve an employee whose last name is ‘Smith’, either as a worker or as a manager of the department that controls the project.

$\text{SMITHS(Essn)} \leftarrow \pi_{\text{Ssn}}\, (\sigma_{\text{Lname='Smith'}}(\text{EMPLOYEE}))$

$\text{SMITH\_WORKER\_PROJS} \leftarrow \pi_{\text{Pno}}(\text{WORKS\_ON} * \text{SMITHS})$

$\text{MGRS} \leftarrow \pi_{\text{Lname, Dnumber}}(\text{EMPLOYEE} \bowtie_{\text{Ssn=Mgr\_ssn}} \text{DEPARTMENT})$

SMITH_MANAGED_DEPTS(Dnum) ← $\pi_{Dnumber}$ ($\sigma_{Lname=‘Smith’}$(MGRS))

SMITH_MGR_PROJS(Pno) ←
$\pi_{Pnumber}$(SMITH_MANAGED_DEPTS * PROJECT)

RESULT ← (SMITH_WORKER_PROJS ∪ SMITH_MGR_PROJS)

In this query, we retrieved the project numbers for projects that involve an employee named Smith as a worker in SMITH_WORKER_PROJS. Then we retrieved the project numbers for projects that involve an employee named Smith as manager of the department that controls the project in SMITH_MGR_PROJS. Finally, we applied the UNION operation on SMITH_WORKER_PROJS and SMITH_MGR_PROJS. As a single in-line expression, this query becomes:

$^{\pi}$Pno $^{(WORKS\_ON}$ $\bowtie$ Essn=Ssn$^{(\pi}$Ssn $^{(\sigma}$Lname=‘Smith’$^{(EMPLOYEE)))}$ ∪ $^{\pi}$Pno

$^{((\pi}$Dnumber $^{(\sigma}$Lname=‘Smith’$^{(\pi}$Lname, Dnumber$^{(EMPLOYEE)))}$

Ssn=Mgr_ssnDEPARTMENT)) $\bowtie$ Dnumber=DnumPROJECT)

Query 5. List the names of all employees with two or more dependents.

Strictly speaking, this query cannot be done in the *basic* (*original*) *relational algebra*. We have to use the AGGREGATE FUNCTION operation with the COUNT aggregate function. We assume that dependents of the *same* employee have *distinct* Dependent_name values.

$T1$(Ssn, No_of_dependents)← $_{Essn}$ $\Im$ $_{COUNT\ Dependent\_name}$(DEPENDENT)

^{T2} ←$^{\sigma}$No_of_dependents>2$^{(T1)}$

RESULT ← π$_{Lname,\ Fname}$$^{(T2}$ * EMPLOYEE)

Query 6. Retrieve the names of employees who have no dependents.

This is an example of the type of query that uses the MINUS (SET DIFFERENCE) operation.

ALL_EMPS ← π_{Ssn}(EMPLOYEE)

EMPS_WITH_DEPS(Ssn) ← π_{Essn}(DEPENDENT)

EMPS_WITHOUT_DEPS ← (ALL_EMPS – EMPS_WITH_DEPS)

RESULT ← $\pi_{Lname,\ Fname}$(EMPS_WITHOUT_DEPS * EMPLOYEE)

We first retrieve a relation with all employee Ssns in ALL_EMPS. Then we create a table with the Ssns of employees who have at least one dependent in EMPS_WITH_DEPS. Then we apply the SET DIFFERENCE operation to retrieve employees Ssns with no dependents in EMPS_WITHOUT_DEPS, and finally join this with EMPLOYEE to retrieve the desired attributes. As a single in-line expression, this query becomes:

$\pi_{\text{Lname, Fname}}((\pi_{\text{Ssn}}(\text{EMPLOYEE}) - \rho_{\text{Ssn}}(\pi_{\text{Essn}}(\text{DEPENDENT}))) * \text{EMPLOYEE})$

Query 7. List the names of managers who have at least one dependent.

$\text{MGRS(Ssn)} \leftarrow \pi_{\text{Mgr\_ssn}}(\text{DEPARTMENT})$

$\text{EMPS\_WITH\_DEPS(Ssn)} \leftarrow \pi_{\text{Essn}}(\text{DEPENDENT})$

$\text{MGRS\_WITH\_DEPS} \leftarrow (\text{MGRS} \cap \text{EMPS\_WITH\_DEPS})$

$\text{RESULT} \leftarrow \pi_{\text{Lname, Fname}}(\text{MGRS\_WITH\_DEPS} * \text{EMPLOYEE})$

In this query, we retrieve the Ssns of managers in MGRS, and the Ssns of employees with at least one dependent in EMPS_WITH_DEPS, then we apply the SET INTERSECTION operation to get the Ssns of managers who have at least one dependent.

As we mentioned earlier, the same query can be specified in many different ways in relational algebra. In particular, the operations can often be applied in various orders. In addition, some operations can be used to replace others; for example, the INTERSECTION operation in Q7 can be replaced by a NATURAL JOIN. As an exercise, try to do each of these sample queries using different operations.