DATABASE MANAGEMENT SYSTEM

MODULE -2

SUBJECT CODE: 22AIM42

FACULTY :K.S.SHASHIKALA

MODULE-2	LOGICAL DESIGN AND RELATIONAL MODEL	22AIM42.1, 22AIM42.3	8 Hours
Domains, Attributes, Tuples, and Relations; Relational Model Constraints; Relational Database Schemas; SQL1: Overview of SQL language; SQL Data Definition and Data Types; Schema change statements in SQL; Enforcing basic constraints in SQL; Basic structure of SQL queries Joins; Logical connectives-AND, OR and NOT; Addition basic operations; Setoperations; Aggregate function			
Case Study	Develop a database for the hospital to maintain the records of various departments, rooms, and doctors in the hospital. It also maintains records of the regular patients, patients admitted in the hospital, the check up of patients done by the doctors, the patients that have been operated, and patients discharged from the hospital.		

Domains, Attributes, Tuples, and Relations-Page no. 61 -SQL-Elmasri & navathe
relational model constraints and relational database schemas,67- 69
SQL data definition and datatypes 89
Schema change statements 137
Specifying constraints
SQL-"Fundamentals of database systems"-Elmasri & navathe-Chapter 4 ,5
Comparisons involving NULL and three-valued logic, 116–117
Select delete update clause 107
Basic structure of SQL queries, joins chapter 4.1 ad 4.2 korth and sudershan
Additional basic operations, set operations page 79,85- korth& sudershan

The Relational data model

- ✓ The **relational data** model was first proposed by **Dr. E.F. Codd** of IBM Research in 1970 in the following paper:
  "A Relational Model for Large Shared Data Banks," Communications of the ACM, June 1970

- ✓ It attracted immediate attention due to its simplicity and mathematical foundation.The above paper caused a major revolution in the field of database management and earned Dr. Codd the coveted **ACM Turing Award**

- ✓ The relational data model uses the concept of a **mathematical relation as its building blocks , which looks somewhat like a table of values.**

- ✓ A Relation is a mathematical concept based on the ideas of **set theory and first-order predicate logic**

The first **commercial implementations** of the relational model became available in the early 1980s, such as the **SQL/DS system** on the MVS operating system by IBM and the Oracle DBMS.

Current popular relational DBMSs (RDBMSs) include:

- ✓ DB2 and Informix Dynamic Server (from IBM)
- ✓ Oracle and Rdb (from Oracle)
- ✓ Sybase DBMS (from Sybase)
- ✓ SQLServer and Access (from Microsoft).

In addition, several open source database systems, such as **MySQL and PostgreSQLare available**

The relational data model concepts

- The relational model represents the database as **a collection of relations**. Informally, each relation resembles **a table of values or a flat file of records. It is called a flat file because each record has a simple linear or flat structure.**
- When a relation is thought of as a table of values, each row in the table represents a **collection of related data values.**
- A row represents a fact that typically corresponds to a real-world entity or relationship.
- The table name and column names are used to help to interpret the meaning of the values in each row. For example, the first table of Figure 1.2 is called STUDENT because each row represents facts about a particular student entity.
- The column names—Name, Student_number, Class, and Major—specify how to interpret the data values in each row, based on the column each value is in.
- All values in a column are of **the same data type.**
- In the formal relational model terminology**, a row is called a tuple, a column header is called an attribute, and the table is called a relation.**
- **The data type describing the types of values that can appear in each column is represented by a domain of possible values**.

STUDENT

Name	Student_number	Class	Major
Smith	17	1	CS
Brown	8	2	CS

COURSE

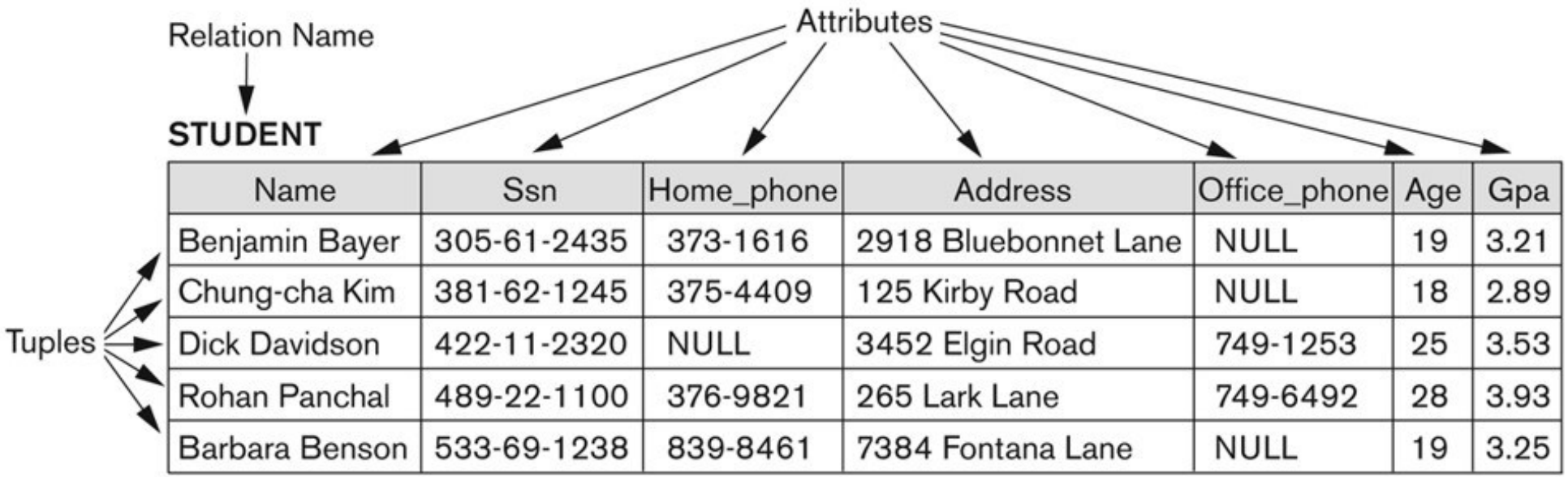
Course_name	Course_number	Credit_hours	Department
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS
Discrete Mathematics	MATH2410	3	MATH
Database	CS3380	3	CS

SECTION

Section_identifier	Course_number	Semester	Year	Instructor
85	MATH2410	Fall	07	King
92	CS1310	Fall	07	Anderson
102	CS3320	Spring	08	Knuth
112	MATH2410	Fall	08	Chang
119	CS1310	Fall	08	Anderson
135	CS3380	Fall	08	Stone

Eg: A database that stores student and course information.

Example of a Relation



Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	NULL	19	3.21
Chung-cha Kim	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
Rohan Panchal	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

Figure 5.1
The attributes and tuples of a relation STUDENT.

- The **Schema** of a Relation(Relation schema):
  - Denoted by R(A1, A2, ...An)
  - R is the **name** of the relation
  - The **attributes** of the relation are A1, A2, ..., An
- Example:

  CUSTOMER (Cust-id, Cust-name, Address, Phone#)
  - CUSTOMER is the relation name
  - Defined over the four attributes: Cust-id, Cust-name, Address, Phone#
- Each attribute has a **domain** or a set of valid values.
  - For example, the domain of Cust-id is 6 digit numbers.

A relation schema is used to describe a relation and R is called the name of this relation.

Domains, Attributes, Tuples, and Relation

Domain

A domain D is **a set of atomic values**. By atomic we mean that each value in the domain is indivisible as far as the formal relational model is concerned.

A common method of specifying a domain is to **specify a data type** from which the data values forming the domain are drawn.

It is also useful to specify a **name for the domain**, to help in interpreting its values.

Some examples of domains follow:

■ **Social_security_numbers** - The set of valid nine-digit Social Security numbers. (This is a unique identifier assigned to each person in the United States for employment, tax, and benefits purposes.)

■ **Name -** The set of character strings that represent names of persons.

■Grade_point_averages- Possible values of computed grade point averages; each must be a real (floating-point) number between 0 and 4.

■ Employee_ages- Possible ages of employees in a company; each must be an integer value between 15 and 80.

■ Academic_department_names - The set of academic department names in a university, such as Computer Science, Economics, and Physics.

Relation schema

A relation schema is used to describe a relation R ,where R is called the name of the relation.

A relation schema R, denoted by R(A1, A2, ..., An), is made up of **a relation name R and a list of attributes, A1, A2, ..., An.**

Each **attribute** Ai is the name of a role played by some domain D in the relation schema R. D is called the domain of Ai and is denoted by dom(Ai).

- Example:

  CUSTOMER (Cust-id, Cust-name, Address, Phone#)
  - CUSTOMER is the relation name
  - Defined over the four attributes: Cust-id, Cust-name, Address, Phone#
- Each attribute has a **domain** or a set of **valid atomic values.**
  - For example, the domain of Cust-id is 6 digit numbers.

DEGREE OF RELATION -EXAMPLE

The **degree (or arity) of a relation** is the **number of attributes** n of its relation schema. A relation of degree seven, which stores information about university students, would contain seven attributes describing each student. as follows:

STUDENT(Name, Ssn, Home_phone, Address, Office_phone, Age, Gpa)

Using the data type of each attribute, the definition is sometimes written as:

STUDENT(Name: string, Ssn: string, Home_phone: string, Address: string, Office_phone: string, Age: integer, Gpa: real)

For this relation schema, STUDENT is the name of the relation, which has seven attributes

Informal Definitions

- RELATION:

Represented by table of values

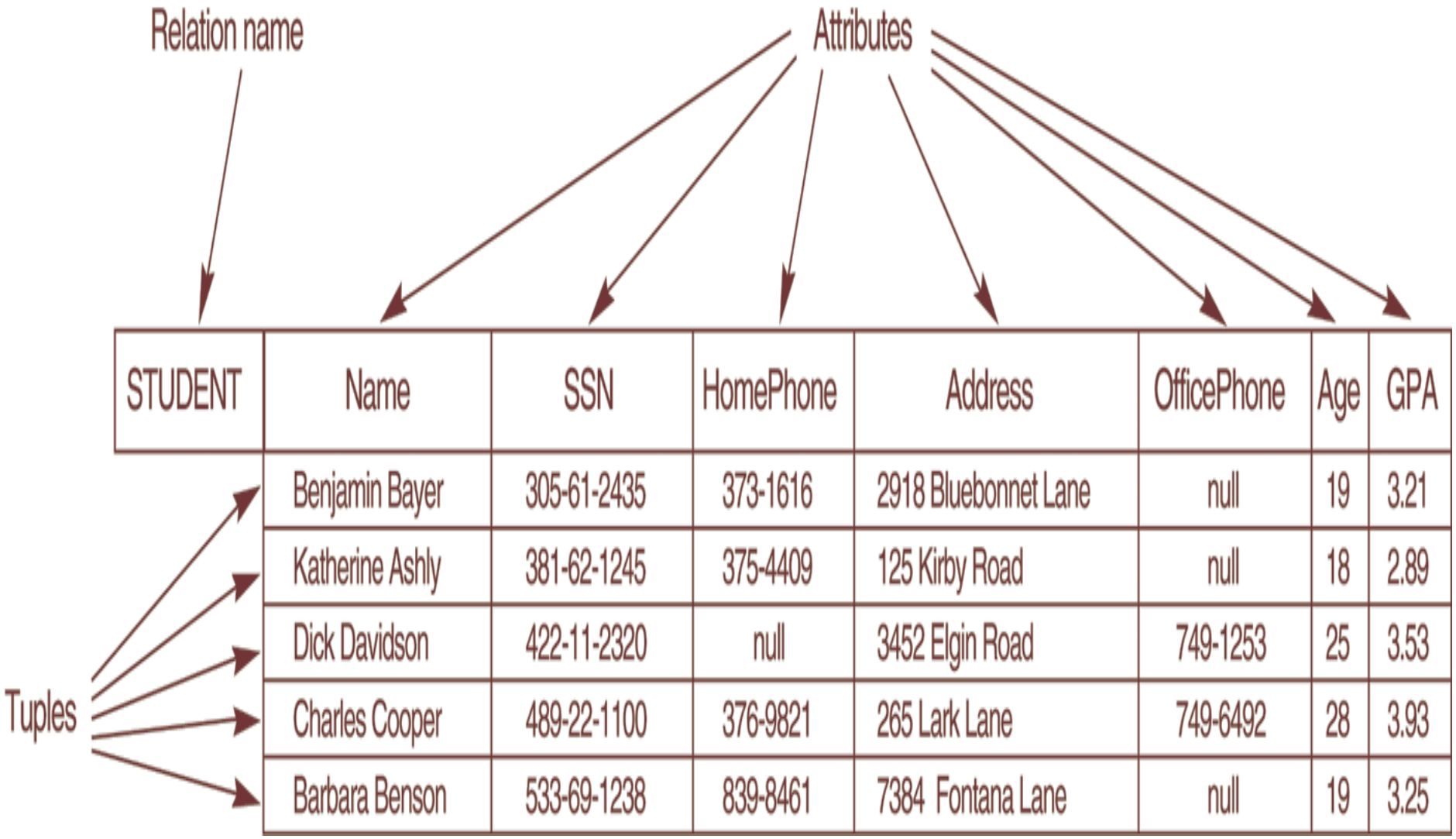
- A relation may be thought of as a **set of rows**.
- Each row represents a fact that corresponds to a real-world **entity** or **relationship**.
- Each row has a value of an **item or set of items** that uniquely identifies that row in the table.
- Each column typically is called by its column name or column header or attribute name.

Informal Definitions-Key of a Relation

- Key of a Relation:
  - Each row has a value of a data item (or set of items) that uniquely identifies that row in the table
    - Called the *key*
  - In the STUDENT table, SSN is the key

  - Sometimes row-ids or sequential numbers are assigned as keys to identify the rows in a table
    - Called artificial key or surrogate key

Example



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

TUPLE

- A row is called a **tuple, which is an ordered set of values.** A relation is a set of such tuples (rows).
- A column header is called an attribute
  - Each attribute value is derived from an appropriate domain.
- The table is called a relation.
  - A relation can be regarded as a ***set of tuples*** (rows).
- The data type describing the types of values an attribute can have is represented by a **domain of possible values.**

- Each row in the CUSTOMER table is a 4-tuple and consists of four values, for example.

<632895, "John Smith", "101 Main St. Atlanta, GA 30332","(404) 894-2000">

Formal Definitions

- The **relation** is formed over the cartesian product of the sets; each set has values from a domain

- The Cartesian product of two sets *A* and *B* is defined to be the set of all pairs (*a, b*) where $a \in A$ and $b \in B$. It is denoted $A \times B$, and is called **the Cartesian product**

An Example of Cartesian Product

STUDENT

SID	SName
111	Williams
222	Johnes

ENROLLMENT

C_NO	SID
1000	111
2000	222

STUDENT × ENROLLMENT

STUDENT

SID	SName
111	Williams
111	Williams
222	Johns
222	Johns

ENROLLMENT

C_NO	SID
1000	111
2000	222
1000	111
2000	222

RESULT

SID	SName	C_NO	SID
111	**Williams**	**1000**	**111**
111	Williams	2000	222
222	Johns	1000	111
222	**Johns**	**2000**	**222**

Relation state

- A relation (or relation state) r(R) is a mathematical relation of degree n on the domains dom(A1), dom(A2), ..., dom(An), which is a subset of the Cartesian product (denoted by ×) of the domains that define R

$$r(R) \subset dom(A1) \times dom(A2) \times \ldots \times dom(An)$$

The Cartesian product specifies all possible combinations of values from the underlying domains.

- The total number of values, in a domain D (assuming that all domains are finite)is represented by **cardinality , or by |D|**
- The total number of tuples in the Cartesian product is:

$$|dom(A1)| \times |dom(A2)| \times \ldots \times |dom(An)|$$

This product of cardinalities of all domains represents the **total number of possible instances or tuples that can ever exist in any relation state r(R)**

Of all these possible combinations, a relation state at a given time is called the **current relation state**—reflects only the **valid tuples that represent a particular state of the real world**

Relation state - Example

- Let R(A1, A2) be a relation schema:
  - Let dom(A1) = {0,1}
  - Let dom(A2) = {a,b,c}
- Then: dom(A1) X dom(A2) is all possible combinations:
  {<0,a> , <0,b> , <0,c>, <1,a>, <1,b>, <1,c> }

- The relation state r(R) $\subset$ dom(A1) X dom(A2)
- For example: r(R) could be {<0,a> , <0,b> , <1,c> }
  - this is one possible state (or "population" or "extension") r of the relation R, defined over A1 and A2.
  - It has three 2-tuples: <0,a> , <0,b> , <1,c>

Shashi KS

Relational Model Constraints and Relational Database Schemas

In a relational database, there will typically be many relations, and the tuples in those relations are usually related in various ways. The state of the whole database will correspond to the states of all its relations at a particular point in time.

There are generally many restrictions or constraints on the actual values in a database state. These constraints are derived from the rules in the miniworld that the database represents.

Various restrictions on data can be specified on a relational database in the form of constraints. Constraints on databases can generally be divided into three main categories:

1. Constraints that are inherent in the data model. (inherent model-based constraints or implicit constraints)
2. Constraints that can be directly expressed in schemas of the data model, typically by specifying them in the DDL (schema-based constraints or explicit constraints)
3. Constraints that cannot be directly expressed in the schemas of the data model, and hence must be expressed and enforced by the application programs.(application-based or semantic constraints or business rules.)

Examples of constraints categories

- ✓ The characteristics of relations, are the inherent constraints of the relational model and belong to the first category. For example, the constraint that a relation cannot have duplicate tuples is an inherent constraint.

- ✓ The second category constraints are the constraints that can be expressed in the schema of the relational model via the DDL.

- ✓ Constraints in the third category are more general, relate to the meaning as well as behavior of attributes, and are difficult to express and enforce within the data model, so they are usually checked within the application programs that perform database updates.

- ✓ Another important category of constraints is data dependencies, which include functional dependencies and multivalued dependencies. They are used mainly for testing the "goodness" of the design of a relational database and are utilized in a process called normalization

Schema-based constraints

Shashi KS

The schema-based constraints include:

Domain constraints, key constraints, constraints on NULLs, entity integrity constraints, and referential integrity constraints

1.Domain Constraints

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

The data types associated with domains typically include standard numeric data types for integers (such as short integer, integer, and long integer) and real numbers (float and double precision float).

Characters, Booleans, fixed-length strings, and variable-length strings are also available, as are date, time, timestamp, and money, or other special data types.

Other possible domains may be described by a subrange of values from a data type or as an enumerated data type in which all possible values are explicitly listed

2.Key Constraints and Constraints on NULL Values

SUPER KEY:

Usually, there are other subsets of attributes of a relation schema R with the property that no two tuples in any relation state r of R should have the same combination of values for these attributes.

Suppose that we denote one such subset of attributes by SK; then for any two distinct tuples t 1 and t 2 in a relation state r of R, we have the constraint that: t 1[SK]≠ t 2[SK]

Any such set of attributes SK is called a superkey of the relation schema R.

A superkey SK specifies a uniqueness constraint that no two distinct tuples in any state r of R can have the same value for SK. Every relation has **at least one default superkey**—the set of all its attributes.

A superkey can have redundant attributes

KEY

Shashi KS

A key K of a relation schema R is a superkey of R with the additional property that removing any attribute A from K leaves a set of attributes K that is not a superkey of R any more. Hence, a key satisfies two properties:

1. Two distinct tuples in any state of the relation cannot have identical values for (all) the attributes in the key. This first property also applies to a superkey.

2. It is a minimal superkey—that is, a superkey from which we cannot remove any attributes and still have the uniqueness constraint in condition 1 hold. This property is not required by a superkey.

A key has no redundancy.

The value of a key attribute can be used to identify uniquely each tuple in the relation.
For example, the Ssn value 305-61-2435 identifies uniquely the tuple corresponding to Benjamin Bayer in the STUDENT relation.

KEY AND SUPER KEY-EXAMPLE

Shashi KS

A set of attributes constituting a key is a property of the relation schema; it is a constraint that should hold on every valid relation state of the schema.

A key is determined from the meaning of the attributes, and **the property is time-invariant**: It must continue to hold when we insert new tuples in the relation

Consider the STUDENT relation of Figure 3.1. The attribute set {Ssn} is a key of STUDENT because no two student tuples can have the same value for Ssn.

Any set of attributes that includes Ssn—for example, {Ssn, Name, Age}—is a superkey. However, the superkey {Ssn, Name, Age} is not a key of STUDENT because removing Name or Age or both from the set still leaves us with a superkey.

In general, any superkey formed from a single attribute is also a key. A key with multiple attributes must require all its attributes together to have the uniqueness property.

CANDIDATE KEY AND PRIMARY KEY

A relation schema may have more than one key. In this case, each of the keys is called a candidate key.

For example, the CAR relation has two candidate keys: License_number and Engine_serial_number.

It is common to designate one of the candidate keys as the primary key of the relation. This is the candidate key whose values are used to identify tuples in the relation.

We use the convention that the attributes that form the primary key of a relation schema are underlined

CAR

License_number	Engine_serial_number	Make	Model	Year
Texas ABC-739	A69352	Ford	Mustang	02
Florida TVP-347	B43696	Oldsmobile	Cutlass	05
New York MPO-22	X83554	Oldsmobile	Delta	01
California 432-TFY	C43742	Mercedes	190-D	99
California RSK-629	Y82935	Toyota	Camry	04
Texas RSK-629	U028365	Jaguar	XJS	04

Figure 3.4
The CAR relation, with two candidate keys: License_number and Engine_serial_number.

3.Relational Databases and Relational Database Schemas

The definitions and constraints we have discussed so far apply to single relations and their attributes.

A relational database usually contains many relations, with tuples in relations that are related in various ways.

A relational database schema S is a set of relation schemas S = {R1, R2, ..., Rm} and a set of integrity constraints IC.

A relational database state10 DB of S is a set of relation states DB = {r1, r2, ..., rm} such that each ri is a state of Ri and such that the ri relation states satisfy the integrity constraints specified in IC.

Figure 3.5 shows a relational database schema that we call COMPANY = {EMPLOYEE, DEPARTMENT, DEPT_LOCATIONS, PROJECT, WORKS_ON, DEPENDENT}.

The underlined attributes represent primary keys. Figure 3.6 shows a relational database state corresponding to the COMPANY schema.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

DEPT_LOCATIONS

Dnumber	Dlocation

PROJECT

Pname	Pnumber	Plocation	Dnum

WORKS_ON

Essn	Pno	Hours

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship

Figure 3.5
Schema diagram for the COMPANY relational database schema.

Figure 3.6
One possible database state for the COMPANY relational database schema.

Shashi KS

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

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

When we refer to a relational database, we implicitly include both its schema and its current state. A database state that does not obey all the integrity constraints is called an invalid state, and a state that satisfies all the constraints in the defined set of integrity constraints IC is called a valid state.

Each relational DBMS must have a data definition language (DDL) for defining a relational database schema. Current relational DBMSs are mostly using SQL for this purpose.

Integrity constraints are specified on a database schema and are expected to hold on every valid database state of that schema. In addition to domain, key, and NOT NULL constraints, two other types of constraints are considered part of the relational model: entity integrity and referential integrity

4.Entity Integrity, Referential Integrity, and Foreign Keys

Shashi KS

The entity integrity constraint states that no primary key value can be NULL. This is because the primary key value is used to identify individual tuples in a relation. Having NULL values for the primary key implies that we cannot identify some tuples. For example, if two or more tuples had NULL for their primary keys, we may not be able to distinguish them if we try to reference them from other relations.

The referential integrity constraint is specified between two relations and is used to maintain the consistency among tuples in the two relations.

Informally, the referential integrity constraint states that a tuple in one relation that refers to another relation must refer to an existing tuple in that relation.
For example, in Figure 3.6, the attribute Dno of EMPLOYEE gives the department number for which each employee works; hence, its value in every EMPLOYEE tuple must match the Dnumber value of some tuple in the DEPARTMENT relation.

Shashi KS

To define referential integrity more formally, first we define the concept of a foreign key. The conditions for a foreign key, given below, specify a referential integrity constraint between the two relation schemas R1 and R2.

A set of attributes FK in relation schema R1 is a foreign key of R1 that references relation R2 if it satisfies the following rules:

1. The attributes in FK have the same domain(s) as the primary key attributes PK of R2; the attributes FK are said to reference or refer to the relation R2.
2. A value of FK in a tuple t 1 of the current state r1(R1) either occurs as a value of PK for some tuple t 2 in the current state r2(R2) or is NULL.

In the former case, we have t1[FK] = t2[PK], and we say that the tuple t 1 references or refers to the tuple t 2. In this definition, R1 is called the referencing relation and R2 is the referenced relation. If these two conditions hold, a referential integrity constraint from R1 to R2 is said to hold. In a database of many relations, there are usually many referential integrity constraints.

Referential integrity constraints-Example

Shashi KS

Referential integrity constraints typically arise from the relationships among the entities represented by the relation schemas. For example, consider the database shown in Figure 3.6.

In the EMPLOYEE relation, the attribute Dno refers to the department for which an employee works; hence, we designate Dno to be a foreign key of EMPLOYEE referencing the DEPARTMENT relation.

This means that a value of Dno in any tuple t 1 of the EMPLOYEE relation must match a value of the primary key of DEPARTMENT—the Dnumber attribute—in some tuple t2 of the DEPARTMENT relation, or the value of Dno can be NULL if the employee does not belong to a department or will be assigned to a department later.

Foreign key can refer to its own relation

A foreign key can refer to its own relation.

For example, the attribute Super_ssn in EMPLOYEE refers to the supervisor of an employee; this is another employee, represented by a tuple in the EMPLOYEE relation.

Hence, Super_ssn is a foreign key that references the EMPLOYEE relation itself.

In Figure 3.6 the tuple for employee ‘John Smith’ references the tuple for employee ‘Franklin Wong,’ indicating that ‘Franklin Wong’ is the supervisor of ‘John Smith.’

We can diagrammatically display referential integrity constraints by drawing a directed arc from each foreign key to the relation it references. For clarity, the arrowhead may point to the primary key of the referenced relation

Schema diagram for Company relational database schema

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

DEPT_LOCATIONS

Dnumber	Dlocation

PROJECT

Pname	Pnumber	Plocation	Dnum

WORKS_ON

Essn	Pno	Hours

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship

Schema diagram with referential integrity constraints

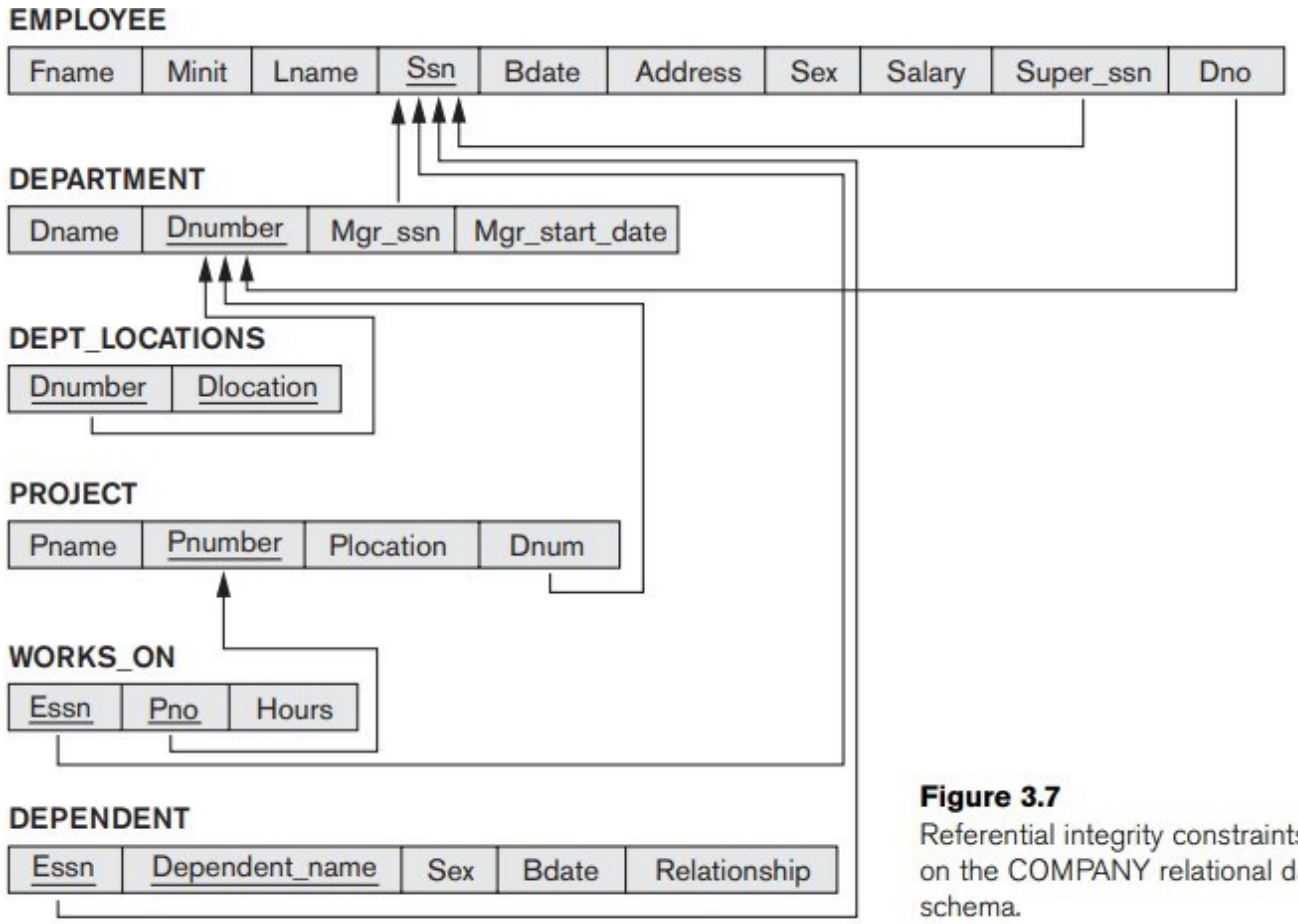


Figure 3.7
Referential integrity constraints displayed on the COMPANY relational database schema.

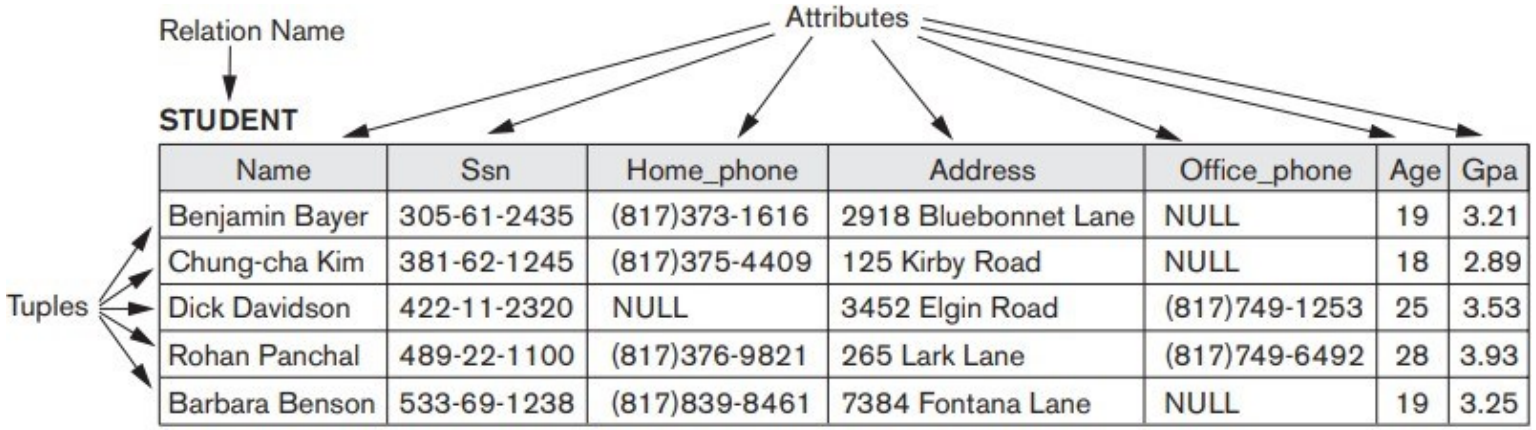
Characteristics of Relations

The characteristics of a relation that make a relation different from a file or a table are:

1.Ordering of Tuples in a Relation:

- A relation is defined as a set of tuples. Mathematically, elements of a set have no order among them; hence, tuples in a relation do not have any particular order. In other words, a relation is **not sensitive to the ordering of tuples**

- The definition of a relation does not specify any order: There is no preference for one ordering over another. Hence, the relation displayed in Figure 3.2 is considered identical to the one shown in Figure 3.1



Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Benjamin Bayer	305-61-2435	(817)373-1616	2918 Bluebonnet Lane	NULL	19	3.21
Chung-cha Kim	381-62-1245	(817)375-4409	125 Kirby Road	NULL	18	2.89
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	(817)749-1253	25	3.53
Rohan Panchal	489-22-1100	(817)376-9821	265 Lark Lane	(817)749-6492	28	3.93
Barbara Benson	533-69-1238	(817)839-8461	7384 Fontana Lane	NULL	19	3.25

Figure 3.1
The attributes and tuples of a relation STUDENT.

Figure 3.2
The relation STUDENT from Figure 3.1 with a different order of tuples.

STUDENT

Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	(817)749-1253	25	3.53
Barbara Benson	533-69-1238	(817)839-8461	7384 Fontana Lane	NULL	19	3.25
Rohan Panchal	489-22-1100	(817)376-9821	265 Lark Lane	(817)749-6492	28	3.93
Chung-cha Kim	381-62-1245	(817)375-4409	125 Kirby Road	NULL	18	2.89
Benjamin Bayer	305-61-2435	(817)373-1616	2918 Bluebonnet Lane	NULL	19	3.21

2.Ordering of Values within a Tuple and an Alternative Definition of a Relation

An alternative definition of a relation can be given, making the **ordering of values in a tuple unnecessary.**

In this definition, a relation schema R = {A1, A2, ..., An} is a set of attributes and a relation state r(R) is a finite set of mappings r = {t1, t2, ..., tm}, where :
Each **tuple ti is a mapping from R to D**
D is the **union (denoted by ∪) of the attribute domains**; that is, **D = dom(A1) ∪ dom(A2) ∪ ... ∪ dom(An).**

t[Ai] must be in dom(Ai) for $1 \leq i \leq n$ for each mapping t in r. Each mapping ti is called a tuple.
According to this definition of tuple as a mapping**, a tuple can be considered as a set of (<attribute>,<value>) pairs, where each pair gives the value of the mapping from an attribute Ai to a value vi from dom(Ai).** The ordering of attributes is not important, because the attribute name appears with its value.

3.Values and NULLs in the Tuples

Each value in a tuple is an **atomic value**; that is, **it is not divisible into components within the framework of the basic relational model**. Hence, composite and multivalued attributes are not allowed. This model is sometimes called the **flat relational model**.

This assumption is called the **first normal form** assumption.

Hence, multivalued attributes must be represented by **separate relations**, and composite attributes are represented only by their **simple component attributes in the basic relational model.**

NULL values

NULL values are used to represent the values of attributes that may be unknown or may not apply to a tuple. A special value, called NULL, is used in these cases.

For example, in Figure 3.1, some STUDENT tuples have NULL for their office phones because they do not have an office (that is, office phone does not apply to these students)

We can have several meanings for NULL values**, such as value unknown, value exists but is not available, or attribute does not apply to this tuple** (also known as **value undefined).**

An example of the last type of NULL will occur if we add an **attribute Visa_status** to the STUDENT relation that applies only to tuples representing foreign students

4.Interpretation (Meaning) of a Relation:

The relation schema can be interpreted as a **declaration or a type of assertion**.

For example, the schema of the STUDENT relation of Figure 3.1 asserts that, in general, a student entity has a Name, Ssn, Home_phone, Address, Office_phone, Age, and Gpa.

Each tuple in the relation can then be interpreted **as a fact or a particular instance of the assertion.**

For example, the first tuple in Figure 3.1 asserts the fact that there is a STUDENT whose Name is Benjamin Bayer, Ssn is 305-61-2435, Age is 19, and so on.

Some relations may represent **facts about entities, whereas other relations may represent facts about relationships**

Overview of SQL Language

- The name SQL is presently expanded as **Structured Query Language.**
- Originally, SQL was called SEQUEL (Structured English QUEry Language) and was designed and implemented at IBM Research as the interface for an experimental relational database system called SYSTEM R.
- SQL is now the standard language for commercial relational DBMSs.
- A joint effort by the American National Standards Institute (ANSI) and the International Standards Organization (ISO) has led to a standard version of SQL (ANSI 1986), called SQL-86 or SQL1.
- A revised and much expanded standard called SQL-92 (also referred to as SQL2) was subsequently developed. The next standard that is well-recognized is SQL:1999, which started out as SQL3.
- Two later updates to the standard are SQL:2003 and SQL:2006, which added XML features among other updates to the language.
- Another update in 2008 incorporated more object database features in SQL

SQL is a **comprehensive database language:** It has statements for **data definitions, queries, and updates. Hence, it is both a DDL and a DML.**

In addition, it has facilities for

- defining views on the database,
- for specifying security and authorization,
- for defining integrity constraints, and
- for specifying transaction controls.
- has rules for embedding SQL statements into a general-purpose programming language such as Java, COBOL, or C/C++.

Eg: Create view

```
CREATE VIEW DetailsView AS
SELECT NAME, ADDRESS
FROM Student_Details
WHERE STU_ID < 4;
```

SQL Data Definition and Data Types

SQL uses the terms table, row, and column for the formal relational model terms **relation, tuple, and attribute, respectively.**

The main SQL command for **data definition is the CREATE statement, which can be used to create schemas, tables (relations), and domains (as well as other constructs such as views, assertions, and triggers).**

i) Schema and Catalog Concepts in SQL

Schema is used to group together tables and other constructs that belong to the same database application.
The concept of an SQL schema was incorporated starting with SQL2 **.**

A database schema, is the way a given database is organized or structured. It organizes the objects in a database

An SQL schema **is identified by a schema name, and includes an authorization identifier to indicate the user or account who owns the schema, as well as descriptors for each element in the schema.**

Schema elements include tables, constraints, views, domains, and other constructs (such as authorization grants) that describe the schema.
A schema is created via the **CREATE SCHEMA statement**, which can include all the schema elements' definitions. Alternatively, the schema can be assigned a **name and authorization identifier,** and the elements can be defined later.

Example:

The following statement creates a schema called COMPANY, owned by the user with authorization identifier 'Jsmith'. Note that each statement in SQL ends with a semicolon.

```
CREATE SCHEMA COMPANY AUTHORIZATION 'Jsmith';
```

In general, not all users are authorized to create schemas and schema elements.

The privilege to **create schemas, tables, and other constructs must be explicitly granted to the relevant user accounts by the system administrator or DBA**

In SQL, Schema and Database concepts are synonymous

```
1 •  create schema mystore;
2 •  create schema mydatabase;
3 •  show databases;
```

Result Grid | Filter Rows: | Export: | Wrap Cell

Database
books
bookss
company
company1
flights
information_schema
mydatabase
mysql
mystore
performance_schema
sha1
shashi1

Catalog

- Catalog is a named collection of schemas in an SQL environment.
- An SQL environment is basically an installation of an SQL-compliant RDBMS on a computer system.
- A catalog always contains a special schema called **INFORMATION_SCHEMA**, which provides information on all the schemas in the catalog and all the element descriptors in these schemas.
- Integrity constraints such as **referential integrity** can be defined between relations only if they exist in **schemas within the same catalog.**
- Schemas within the same catalog can also share certain elements, **such as domain definitions**

SQL DDL commands for creating tables

- ✓ The CREATE TABLE command is used to specify a new relation by giving it a name and specifying its attributes and initial constraints.
- ✓ The attributes are specified first, and each attribute is given a name, a data type to specify its domain of values, and any attribute constraints, such as NOT NULL.
- ✓ The key, entity integrity, and referential integrity constraints can be specified within the CREATE TABLE statement after the attributes are declared, or they can be added later using the ALTER TABLE command

```
CREATE TABLE EMPLOYEE
        ( Fname          VARCHAR(15)        NOT NULL,
          Minit          CHAR,
          Lname          VARCHAR(15)        NOT NULL,
          Ssn            CHAR(9)            NOT NULL,
          Bdate          DATE,
          Address        VARCHAR(30),
          Sex            CHAR,
          Salary         DECIMAL(10,2),
          Super_ssn      CHAR(9),
          Dno            INT                NOT NULL,
        PRIMARY KEY (Ssn),
        FOREIGN KEY (Super_ssn) REFERENCES EMPLOYEE(Ssn),
        FOREIGN KEY (Dno) REFERENCES DEPARTMENT(Dnumber) );
```

Figure 4.1 - SQL CREATE TABLE data definition statements for defining the COMPANY schema

```
CREATE TABLE DEPARTMENT
        ( Dname                 VARCHAR(15)             NOT NULL,
          Dnumber               INT                     NOT NULL,
          Mgr_ssn               CHAR(9)                 NOT NULL,
          Mgr_start_date        DATE,
        PRIMARY KEY (Dnumber),
        UNIQUE (Dname),
        FOREIGN KEY (Mgr_ssn) REFERENCES EMPLOYEE(Ssn) );
CREATE TABLE DEPT_LOCATIONS
        ( Dnumber               INT                     NOT NULL,
          Dlocation             VARCHAR(15)             NOT NULL,
        PRIMARY KEY (Dnumber, Dlocation),
        FOREIGN KEY (Dnumber) REFERENCES DEPARTMENT(Dnumber) );
```

SQL CREATE TABLE data definition statements for defining the COMPANY schema

```
CREATE TABLE PROJECT
        ( Pname                 VARCHAR(15)             NOT NULL,
          Pnumber               INT                     NOT NULL,
          Plocation             VARCHAR(15),
          Dnum                  INT                     NOT NULL,
        PRIMARY KEY (Pnumber),
        UNIQUE (Pname),
        FOREIGN KEY (Dnum) REFERENCES DEPARTMENT(Dnumber) );
CREATE TABLE WORKS_ON
        ( Essn                  CHAR(9)                 NOT NULL,
          Pno                   INT                     NOT NULL,
          Hours                 DECIMAL(3,1)            NOT NULL,
        PRIMARY KEY (Essn, Pno),
        FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn),
        FOREIGN KEY (Pno) REFERENCES PROJECT(Pnumber) );
CREATE TABLE DEPENDENT
        ( Essn                  CHAR(9)                 NOT NULL,
          Dependent_name        VARCHAR(15)             NOT NULL,
          Sex                   CHAR,
          Bdate                 DATE,
          Relationship          VARCHAR(8),
        PRIMARY KEY (Essn, Dependent_name),
        FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn) );
```

SQL CREATE TABLE data definition statements for defining the COMPANY schema

Base and virtual relations

- The relations declared through CREATE TABLE statements are called **base tables (or base relations);** this means that the relation and its tuples are actually created and stored as a file by the DBMS.
- The relations declared through **CREATE VIEW** statements are called **virtual relations,** which may or may not correspond to an actual physical file.
- In SQL, the attributes in a base table are considered to be ordered in the sequence in which they are specified in the CREATE TABLE statement.
- However, **rows (tuples) are not considered to be ordered within a relation**

Foreign keys that are specified either via circular references or those that refer to a table that has not yet been created may cause errors. For example, the foreign key Super_ssn in the EMPLOYEE table is a circular reference because it refers to the table itself.

The foreign key Dno in the EMPLOYEE table refers to the DEPARTMENT table, which has not been created yet**. To deal with this type of problem, these constraints can be left out of the initial CREATE TABLE statement, and then added later using the ALTER TABLE statement**

The ALTER Command

The definition of a base table or of other named schema elements can be changed by using the ALTER command.

For base tables, the possible alter table actions include adding or dropping a column (attribute), changing a column definition, and adding or dropping table constraints.

For example, to add an attribute for keeping track of jobs of employees to the EMPLOYEE base relation in the COMPANY schema (see Figure 4.1), we can use the command

ALTER TABLE COMPANY.EMPLOYEE ADD COLUMN Job VARCHAR(12);

We must still enter a value for the new attribute Job for each individual EMPLOYEE tuple.

This can be done **either by specifying a default clause or by using the UPDATE command individually on each tuple**

If no default clause is specified, the new attribute will have NULLs in all the tuples of the relation immediately after the command is executed; hence, the NOT NULL constraint is not allowed in this case

Attribute Data Types and Domains in SQL

The basic data types available for attributes include **numeric, character string, bit string, Boolean, date, and time.**

Numeric data types include integer numbers of various sizes (INTEGER or INT, and SMALLINT) and floating-point (real) numbers of various precision (FLOAT or REAL, and DOUBLE PRECISION).

Formatted numbers can be declared by using DECIMAL(i,j)—or DEC(i,j) or NUMERIC(i,j)—where i, the precision, is the total number of decimal digits and j, the scale, is the number of digits after the decimal point. The default for scale is zero, and the default for precision is implementation-defined.

■Character-string data types are either fixed length—**CHAR(n)** or CHARACTER(n), where n is the number of characters—or varying length— VARCHAR(n) or CHAR VARYING(n) or CHARACTER VARYING(n), where n is the maximum number of characters

Another variable-length string data type called CHARACTER LARGE OBJECT or CLOB is also available to specify columns that have large text values, such as documents.

The CLOB maximum length can be specified in kilobytes (K), megabytes (M), or gigabytes (G). For example, CLOB(20M) specifies a maximum length of 20 megabytes.

Bit-string data types are **either of fixed length n—BIT(n)—or varying length—BIT VARYING**(n), where n is the maximum number of bits. The default for n, the length of a character string or bit string, is 1

- A Boolean data type has the traditional values of TRUE or FALSE. In SQL, because of the presence of NULL values, a three-valued logic is used, so a third possible value for a Boolean data type is **UNKNOWN.**

- The DATE data type has ten positions, and its components are YEAR, MONTH, and DAY in the form **YYYY-MM-DD**.

- The TIME data type has at least eight positions, with the components HOUR, MINUTE, and SECOND in the form HH:MM:SS

A timestamp data type (TIMESTAMP) includes the DATE and TIME fields, plus a minimum of six positions for decimal fractions of seconds and an optional WITH TIME ZONE qualifier.

Literal values are represented by single quoted strings preceded by the keyword TIMESTAMP, with a blank space between data and time; for example, TIMESTAMP '2008-09-27 09:12:47.648302'.

Schema Change Statements in SQL

The schema evolution commands available in SQL can be used to alter a schema by adding or dropping tables, attributes, constraints, and other schema elements.
This can be done while the database is operational and does not require recompilation of the database schema. Certain checks must be done by the DBMS to ensure that the changes do not affect the rest of the database and make it inconsistent.

1.The DROP Command
The DROP command can be used to drop named schema elements, such as tables, domains, or constraints. One can also drop a schema. For example, if a whole schema is no longer needed, the DROP SCHEMA command can be used.
There are two drop behavior options: CASCADE and RESTRICT. For example, to remove the COMPANY database schema and all its tables, domains, and other elements, the CASCADE option is used as follows:

DROP SCHEMA COMPANY CASCADE;
If the RESTRICT option is chosen in place of CASCADE, the schema is dropped only if it has no elements in it; otherwise, the DROP command will not be executed.
To use the RESTRICT option, the user must first individually drop each element in the schema, then drop the schema itself

If a base relation within a schema is no longer needed, the relation and its definition can be deleted by using the DROP TABLE command. For example, if we no longer wish to keep track of dependents of employees in the COMPANY database of Figure 4.1, we can get rid of the DEPENDENT relation by issuing the following command:

DROP TABLE DEPENDENT CASCADE;

If the RESTRICT option is chosen instead of CASCADE, a table is dropped only if it is not referenced in any constraints (for example, by foreign key definitions in another relation) or views (see Section 5.3) or by any other elements.

The ALTER Command

The definition of a base table or of other named schema elements can be changed by using the ALTER command.

For base tables, the possible alter table actions include **adding or dropping a column (attribute), changing a column definition, and adding or dropping table constraints.**

For example, to add an attribute for keeping track of jobs of employees to the EMPLOYEE base relation in the COMPANY schema , we can use the command

ALTER TABLE COMPANY.EMPLOYEE ADD COLUMN Job VARCHAR(12);

We must still enter a value for the new attribute Job for each individual EMPLOYEE tuple. This can be done either by specifying a default clause or by using the UPDATE command individually on each tuple.

If no default clause is specified, the new attribute will have NULLs in all the tuples of the relation immediately after the command is executed; hence, the NOT NULL constraint is not allowed in this case.

To drop a column, we must choose either CASCADE or RESTRICT for drop behavior.

If CASCADE is chosen, all constraints and views that reference the column are dropped automatically from the schema, along with the column.

If RESTRICT is chosen, the command is successful only if no views or constraints (or other schema elements) reference the column.

For example, the following command removes the attribute Address from the EMPLOYEE base table:

ALTER TABLE COMPANY.EMPLOYEE DROP COLUMN Address CASCADE;

It is also possible to alter a column definition by dropping an existing default clause or by defining a new default clause. The following examples illustrate this clause:

ALTER TABLE COMPANY.DEPARTMENT ALTER COLUMN Mgr_ssn DROP DEFAULT; ALTER TABLE COMPANY.DEPARTMENT ALTER COLUMN Mgr_ssn SET DEFAULT '333445555';

Adding or dropping a named constraint:

One can also change the constraints specified on a table by adding or dropping a named constraint.

To be dropped, a constraint must have been given a name when it was specified.

For example, to drop the constraint named EMPSUPERFK from the EMPLOYEE relation, we write:

ALTER TABLE COMPANY.EMPLOYEE DROP CONSTRAINT EMPSUPERFK CASCADE;

Once this is done, we can redefine a replacement constraint by **adding a new constraint** to the relation, if needed.

This is specified by using the ADD keyword in the ALTER TABLE statement followed by the new constraint, which can be named or unnamed and can be of any of the table constraint types

Syntax for adding the unique key constraint to single as well as multiple columns is given below:

Syntax :

Adding unique key constraint to a column:

ALTER TABLE <table_name> ADD UNIQUE (<column_name>);

Adding unique key constraint to multiple columns:

ALTER TABLE <table_name>

ADD CONSTRAINT <identifier_name> UNIQUE (<column_name1>,<column_name2>,…);

Summary of SQL Syntax

```
CREATE TABLE <table name> ( <column name> <column type> [ <attribute constraint> ]
                            { , <column name> <column type> [ <attribute constraint> ] }
                            [ <table constraint> { , <table constraint> } ] )

DROP TABLE <table name>
ALTER TABLE <table name> ADD <column name> <column type>

SELECT [ DISTINCT ] <attribute list>
FROM ( <table name> { <alias> } | <joined table> ) { , ( <table name> { <alias> } | <joined table> ) }
[ WHERE <condition> ]
[ GROUP BY <grouping attributes> [ HAVING <group selection condition> ] ]
[ ORDER BY <column name> [ <order> ] { , <column name> [ <order> ] } ]

<attribute list> ::= ( * | ( <column name> | <function> ( ( [ DISTINCT ] <column name> | * ) ) )
                       { , ( <column name> | <function> ( ( [ DISTINCT] <column name> | * ) ) } ) )

<grouping attributes> ::= <column name> { , <column name> }

<order> ::= ( ASC | DESC )

INSERT INTO <table name> [ ( <column name> { , <column name> } ) ]
( VALUES ( <constant value> , { <constant value> } ) { , ( <constant value> { , <constant value> } ) }
| <select statement> )

DELETE FROM <table name>
[ WHERE <selection condition> ]

UPDATE <table name>
SET <column name> = <value expression> { , <column name> = <value expression> }
[ WHERE <selection condition> ]

CREATE [ UNIQUE] INDEX <index name>
ON <table name> ( <column name> [ <order> ] { , <column name> [ <order> ] } )
[ CLUSTER ]

DROP INDEX <index name>

CREATE VIEW <view name> [ ( <column name> { , <column name> } ) ]
AS <select statement>

DROP VIEW <view name>
```

NOTE: The commands for creating and dropping indexes are not part of standard SQL.

Enforcing Basic constraints in SQL

The basic constraints that can be specified in SQL as part of table creation include key and referential integrity constraints, restrictions on attribute domains and NULLs, and constraints on individual tuples within a relation.

Specifying Attribute Constraints and Attribute Defaults

Because SQL allows NULLs as attribute values, a constraint NOT NULL may be specified if NULL is not permitted for a particular attribute.

This is always implicitly specified for the attributes that are part of the primary key of each relation, but it can be specified for any other attributes whose values are required not to be NULL,

Specifying Attribute Defaults

It is also possible to define a default value for an attribute by appending the clause DEFAULT to an attribute definition.

The default value is included in any new tuple if an explicit value is not provided for that attribute.
If no default clause is specified, the default **default value is NULL for attributes that do not have the NOT NULL constraint.**

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255) DEFAULT   'Bangalore '
);
```

The CHECK constraint

- The CHECK constraint is used to limit the value range that can be placed in a column.
- If you define a CHECK constraint on a column it will allow only certain values for this column.
- If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

The following SQL creates a CHECK constraint on the "Age" column when the "Persons" table is created. The CHECK constraint ensures that the age of a person must be 18, or older:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CHECK (Age>=18)
);
```

Specifying Key and Referential Integrity Constraints

There are special clauses within the CREATE TABLE statement to specify the keys and referential integrity constraints

The **PRIMARY KEY** clause specifies one or more attributes that make up the primary key of a relation.

If a primary key has a single attribute, the clause can follow the attribute directly.

For example, the primary key of DEPARTMENT can be specified as follows :

```
Dnumber INT PRIMARY KEY;
```

The UNIQUE clause specifies alternate (secondary) keys, as illustrated in the DEPARTMENT and PROJECT table declarations . The UNIQUE clause can also be specified directly for a secondary key if the secondary key is a single attribute:

```
Dname VARCHAR(15) UNIQUE;
```

FOREIGN KEY Constraint

The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.

A FOREIGN KEY is a **field (or collection of fields) in one table, that refers to the PRIMARY KEY in another table.**

The table with the foreign key is called the **child table, and the table with the primary key is called the referenced or parent table.**

Persons Table

PersonID	LastName	FirstName	Age
1	Hansen	Ola	30
2	Svendson	Tove	23
3	Pettersen	Kari	20

Orders Table

OrderID	OrderNumber	PersonID
1	77895	3
2	44678	3
3	22456	2
4	24562	1

The "PersonID" column in the "Orders" table points to the "PersonID" column in the "Persons" table.

The "PersonID" column in the "Persons" table is the PRIMARY KEY in the "Persons" table.

The "PersonID" column in the "Orders" table is a FOREIGN KEY in the "Orders" table.

The FOREIGN KEY constraint **prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the parent table.**

SQL code to create a FOREIGN KEY on the "PersonID" column when the "Orders" table is created:

```
CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);
```

A referential integrity constraint can be violated when tuples are inserted or deleted, or when a foreign key or primary key attribute value is modified. The default action that SQL takes for an integrity violation is to reject the update operation that will cause a violation, which is known as the RESTRICT option.

However, the schema designer can specify an alternative action to be taken by attaching a referential triggered action clause to any foreign key constraint.

ON DELETE CASCADE clause in MySQL is used to automatically **remove** the matching records from the child table when we delete the rows from the parent table. It is a kind of referential action related to the **foreign key**.

MySQL ON DELETE CASCADE Example

First, we are going to create two tables named **Employee and Payment**. Both tables are related through a foreign key with on delete cascade operation. Here, an Employee is the **parent table**, and Payment is the **child table**. The following scripts create both tables along with their records.

Table: Employee

The following statement creates a table Employee:

```
1.CREATE TABLE Employee (
2.  emp_id int(10) NOT NULL,
3.  name varchar(40) NOT NULL,
4.  birthdate date NOT NULL,
5.  gender varchar(10) NOT NULL,
6.  hire_date date NOT NULL,
7.  PRIMARY KEY (emp_id)
8.);
```

Next, execute the insert query to fill the records.

```
1.INSERT INTO Employee (emp_id, name, birthdate, gender, hire_date) VALUES
2.(101, 'Bryan', '1988-08-12', 'M', '2015-08-26'),
3.(102, 'Joseph', '1978-05-12', 'M', '2014-10-21'),
4.(103, 'Mike', '1984-10-13', 'M', '2017-10-28'),
```

creates a table Payment:

```
1.CREATE TABLE Payment (
2.  payment_id int(10) PRIMARY KEY NOT NULL,
3.  emp_id int(10) NOT NULL,
4.  amount float NOT NULL,
5.  payment_date date NOT NULL,
6.  FOREIGN KEY (emp_id) REFERENCES Employee (emp_id) ON DELETE CASCADE
7.);
```

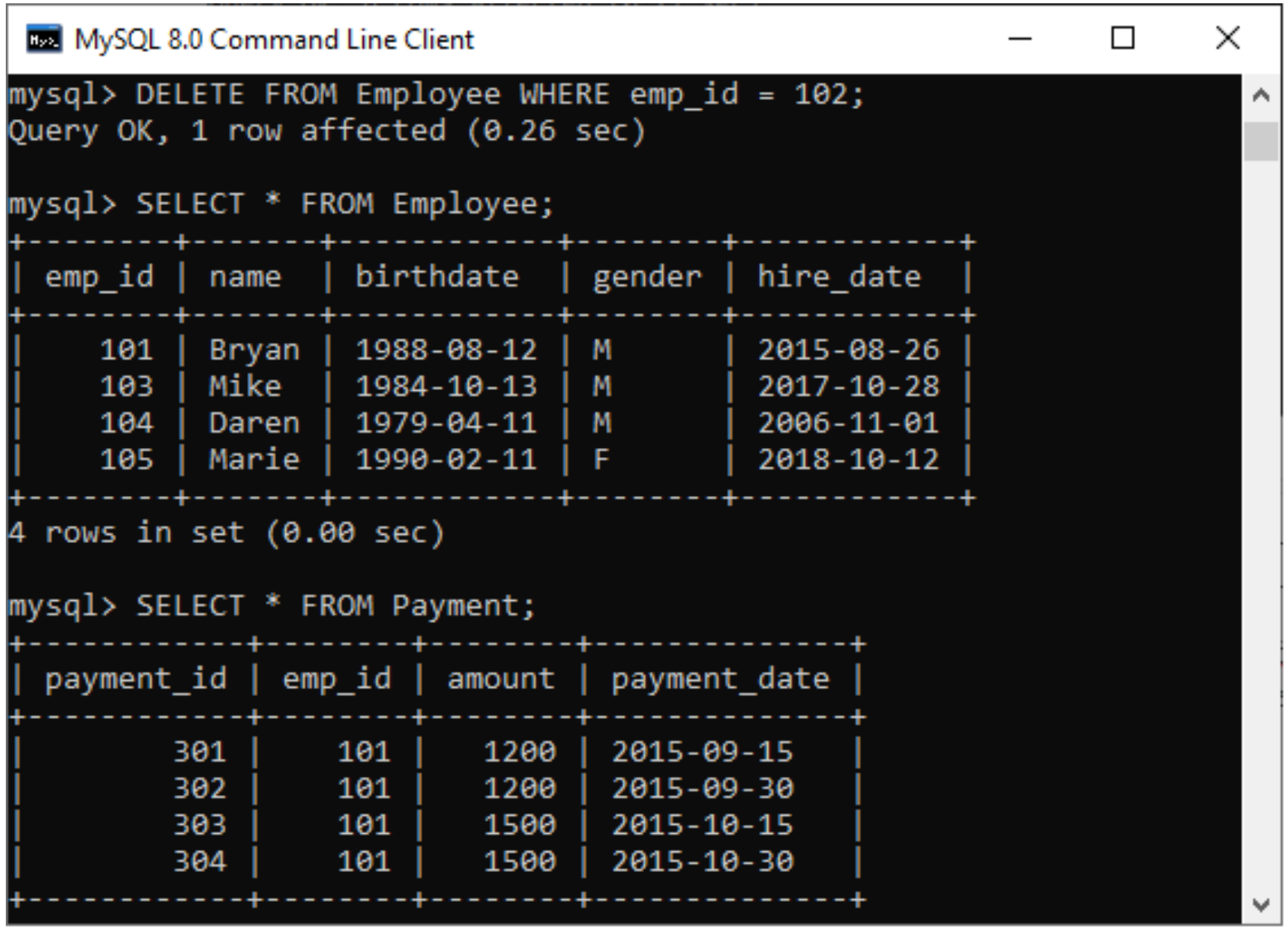
Next, execute the insert statement to fill the records into a table.

```
1.INSERT INTO Payment (payment_id, emp_id, amount, payment_date) VALUES
2.(301, 101, 1200, '2015-09-15'),
3.(302, 101, 1200, '2015-09-30'),
4.(303, 101, 1500, '2015-10-15'),
```

delete data from the parent table Employee. To do this, execute the following statement:

```
1.mysql> DELETE FROM Employee WHERE emp_id = 102;
```

The above statement will delete the employee records whose **emp_id = 102** and **referencing** data into the child table. We can verify the data using the SELECT statement that will give the following output:



In the above output, we can see that all the rows referencing to emp_id = 102 were automatically deleted from both tables.

The following constraints are commonly used in SQL:

- `NOT NULL` - Ensures that a column cannot have a NULL value
- `UNIQUE` - Ensures that all values in a column are different
- `PRIMARY KEY` - A combination of a `NOT NULL` and `UNIQUE`. Uniquely identifies each row in a table
- `FOREIGN KEY` - Prevents actions that would destroy links between tables
- `CHECK` - Ensures that the values in a column satisfies a specific condition
- `DEFAULT` - Sets a default value for a column if no value is specified
- `CREATE INDEX` - Used to create and retrieve data from the database very quickly

Joins in SQL

JOIN in its simplest form can be thought of as a means to retrieve/update or delete data from multiple tables against a single query.
So, in essence, JOIN combines 2 or more tables to fetch data against a given condition.

MySQL JOIN is used to fetch, update or delete data from 2 or more tables against a given condition.

Hence, JOIN is always used in conjunction with SELECT, UPDATE, or DELETE statements

Syntax of JOIN with SELECT command:

SELECT {column_list} FROM **tableName1 {joinType} tableName2**
ON {joinCondition}

The different parts of the syntax include:

{column_list} – This represents the names of columns we want to retrieve as the result of our query.
{JoinType} – This indicates the type of Join that we are applying.
There are following different types of JOINS that can fetch data:

- INNER JOIN
- OUTER JOIN
- LEFT OUTER JOIN
- RIGHT OUTER JOIN
- CROSS JOIN

{JoinCondition} – This is the column conditions which would be used for the JOIN to query and fetch data.

Different types of the JOINs in SQL:

1. (INNER) JOIN: Returns records that have matching values in both tables
2. LEFT (OUTER) JOIN: Returns all records from the left table, and the matched records from the right table
3. RIGHT (OUTER) JOIN: Returns all records from the right table, and the matched records from the left table
4. FULL (OUTER) JOIN: Returns all records when there is a match in either left or right table

```
create database sqljoins;
use sqljoins;

CREATE TABLE Employee_Department (
 name varchar(100),
  id INT NOT NULL auto_increment,
PRIMARY KEY (id));

CREATE TABLE Employee (
 name varchar(100),
 id int not null auto_increment,
 address varchar(100),
 Department_id int,
PRIMARY KEY (id),
FOREIGN KEY (Department_id)      references
Employee_Department(id));
```

```
INSERT INTO `Employee_Department`
VALUES ('Information Technology','1'),
('HR','2'),
('Finance','3'),
('Accounting','4'),
('Housekeeping','5'),
('Security','6'),
('Support','7'),
('Contract Staff','8');

INSERT INTO `Employee`
VALUES ('Veniam','1','640 Damon Junction\nEast Mathew, NY 68818','3'),
('Molestiae','2','6658 Hollis Club\nErnamouth, TX 19743','4'),
('Officiis','3','59965 Mason Neck Apt. 985\nKareemborough, NV 85535','5'),
('Rerum','4','91067 Geovany Fort\nHanefort, WA 92863','6'),
('Et','5','7647 Reva Shores Suite 970\nNew Audrafort, OH 17846-5397','2'),
('Et','6','9419 Carmela Burg Apt. 687\nAimeebury, SD 32389-4489','8'),
('Laborum','7','6961 Weissnat Drive\nDonnellyfort, MT 53947','6'),
('Cupiditate','8','117 Nellie Summit Suite 982\nSouth Heavenfurt, CA 45675','8');
```

```
41 •  select * from Employee_Department;
```

Result Grid | Filter Rows: | Edit:

name	id
Information Technology	1
HR	2
Finance	3
Accounting	4
Housekeeping	5
Security	6
Support	7
Contract Staff	8

Employee_Department table

```
40 •  select * from Employee;
```

Result Grid | Filter Rows: | Edit:

name	id	address	Department_id
Veniam	1	640 Damon Junction East Math...	3
Molestiae	2	6658 Hollis Club Ernamouth, TX...	4
Officiis	3	59965 Mason Neck Apt. 985 Ka...	5
Rerum	4	91067 Geovany Fort Hanefort,...	6
Et	5	7647 Reva Shores Suite 970 N...	2
Et	6	9419 Carmela Burg Apt. 687 Ai...	8
Laborum	7	6961 Weissnat Drive Donnellyf...	6
Cupiditate	8	117 Nellie Summit Suite 982 So...	8

Employee table

Inner join query:

```
SELECT Employee.name as Employee_name, Employee_Department.name
as Department_name
FROM Employee INNER JOIN Employee_Department
ON Employee.Department_id = Employee_Department.id;
```

Query result:

Result Grid | Filter Rows:

Employee_name	Department_name
Veniam	Finance
Molestiae	Accounting
Officiis	Housekeeping
Rerum	Security
Et	HR
Et	Contract Staff
Laborum	Security
Cupiditate	Contract Staff

Here we have used column name **aliases.**
Example: Employee.name as Employee_name just to make the results more readable and comprehensive.

Without using AS keyword in Select query

```
35 •    SELECT Employee.name , Employee_Department.name
36      FROM Employee
37      INNER JOIN Employee_Department
38      ON Employee.Department_id = Employee_Department.id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

	name	name
▶	Veniam	Finance
	Molestiae	Accounting
	Officiis	Housekeeping
	Rerum	Security
	Et	HR
	Et	Contract Staff
	Laborum	Security
	Cupiditate	Contract Staff

Modified query to fetch just the names beginning with the letter ‘e’.

```
SELECT Employee.name as Employee_name, Employee_Department.name as Department_name
FROM Employee  INNER JOIN Employee_Department
ON Employee.Department_id = Employee_Department.id
WHERE Employee.name like 'e%'
```

OUTER JOIN

OUTER JOIN is used to fetch data from 2 or more tables with an exception of including unmatched rows (or rows having null data for the queried columns) as well.

There are 2 types of OUTER JOINs

a) LEFT OUTER JOIN

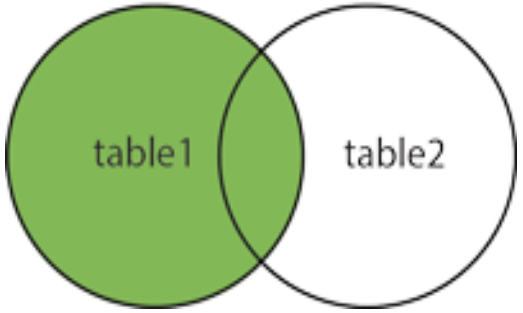
This type of Join would fetch all the rows (including NULL values) from the table which is on the left side of the JOIN query. Returns all records from the left table, and the matched records from the right table

b) RIGHT OUTER JOIN

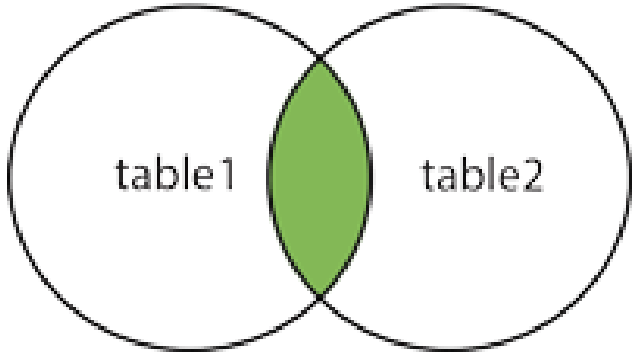
Similar to LEFT JOIN, in this type of Join all the records that do not match from the right table are returned with NULL values on the columns for the left side table. Returns all records from the right table, and the matched records from the left table

In the OUTER JOIN queries – the RIGHT OUTER JOIN and LEFT OUTER JOIN could be just specified as RIGHT JOIN and LEFT JOIN respectively for more readability.

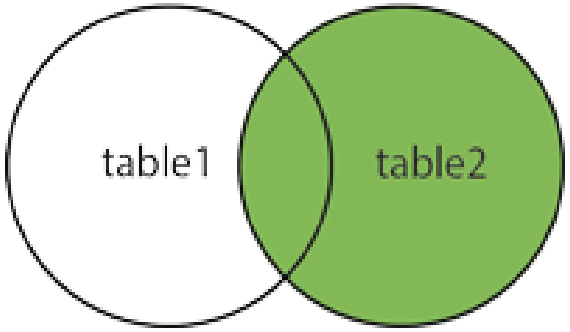
LEFT JOIN



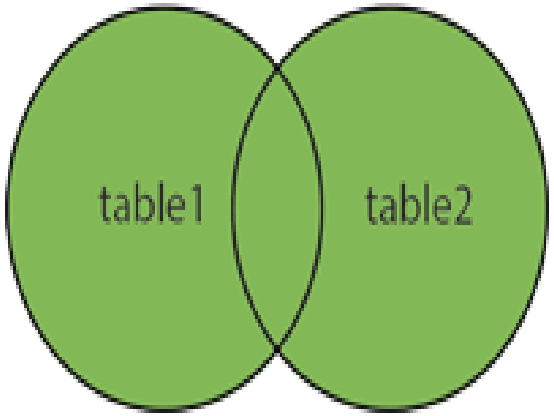
INNER JOIN



RIGHT JOIN



FULL OUTER JOIN



Outer Join Example

Create a new table office_locations having fields – id and address
Add a new column named `office_id` to the originally created Employee table.

Queries:

```
CREATE TABLE office_locations (     address varchar(100), id INT NOT NULL
auto_increment,     PRIMARY KEY (id));

ALTER TABLE Employee ADD COLUMN officeid int;

INSERT INTO office_locations(address)
VALUES('Bangalore'),('Mumbai'),('Seattle'),('Santa Clara');

select * from employee;
set sql_safe_updates=0;
UPDATE Employee SET officeid=1 where id % 2 = 0;
UPDATE Employee SET officeid=2 where id % 2 <> 0;
```

```
73 •    select * from Employee;
```

Result Grid | Filter Rows: | Edit: | Export/

name	id	address	Department_id	officeid
Veniam	1	640 Damon Junction East Math...	3	2
Molestiae	2	6658 Hollis Club Ernamouth, TX...	4	1
Officiis	3	59965 Mason Neck Apt. 985 Ka...	5	2
Rerum	4	91067 Geovany Fort Hanefort,...	6	1
Et	5	7647 Reva Shores Suite 970 N...	2	2
Et	6	9419 Carmela Burg Apt. 687 Ai...	8	1
Laborum	7	6961 Weissnat Drive Donnellyf...	6	2
Cupiditate	8	117 Nellie Summit Suite 982 So...	8	1

Employee

office_locations

```
74 •    select * from office_locations;
```

Result Grid | Filter Rows:

address	id
Bangalore	1
Mumbai	2
Seattle	3
Santa Clara	4

For example, SQL Query to find out what is the office address of all the Employees.

```
SELECT * from Employee
LEFT OUTER JOIN office_locations
ON Employee.office_id = office_locations.id
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

name	id	address	Department_id	officeid	address	id
Veniam	1	640 Damon Junction East Math...	3	2	Mumbai	2
Molestiae	2	6658 Hollis Club Ernamouth, TX...	4	1	Bangalore	1
Officiis	3	59965 Mason Neck Apt. 985 Ka...	5	2	Mumbai	2
Rerum	4	91067 Geovany Fort Hanefort,...	6	1	Bangalore	1
Et	5	7647 Reva Shores Suite 970 N...	2	2	Mumbai	2
Et	6	9419 Carmela Burg Apt. 687 Ai...	8	1	Bangalore	1
Laborum	7	6961 Weissnat Drive Donnellyf...	6	2	Mumbai	2
Cupiditate	8	117 Nellie Summit Suite 982 So...	8	1	Bangalore	1

RIGHT OUTER JOIN

SQL Query to find out the Employees of **all the** office locations

```
78 •    SELECT * from Employee
79      right OUTER JOIN office_locations
80      ON Employee.officeid = office_locations.id;
81
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

name	id	address	Department_id	officeid	address	id
Cupiditate	8	117 Nellie Summit Suite 982 So...	8	1	Bangalore	1
Et	6	9419 Carmela Burg Apt. 687 Ai...	8	1	Bangalore	1
Rerum	4	91067 Geovany Fort Hanefort,...	6	1	Bangalore	1
Molestiae	2	6658 Hollis Club Ernamouth, TX...	4	1	Bangalore	1
Laborum	7	6961 Weissnat Drive Donnellyf...	6	2	Mumbai	2
Et	5	7647 Reva Shores Suite 970 N...	2	2	Mumbai	2
Officiis	3	59965 Mason Neck Apt. 985 Ka...	5	2	Mumbai	2
Veniam	1	640 Damon Junction East Math...	3	2	Mumbai	2
NULL	NULL	NULL	NULL	NULL	Seattle	3
NULL	NULL	NULL	NULL	NULL	Santa Clara	4

```
81 •    SELECT * from Employee
82      cross JOIN office_locations
83      ON Employee.officeid = office_locations.id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

name	id	address	Department_id	officeid	address	id
Veniam	1	640 Damon Junction East Math...	3	2	Mumbai	2
Molestiae	2	6658 Hollis Club Ernamouth, TX...	4	1	Bangalore	1
Officiis	3	59965 Mason Neck Apt. 985 Ka...	5	2	Mumbai	2
Rerum	4	91067 Geovany Fort Hanefort,...	6	1	Bangalore	1
Et	5	7647 Reva Shores Suite 970 N...	2	2	Mumbai	2
Et	6	9419 Carmela Burg Apt. 687 Ai...	8	1	Bangalore	1
Laborum	7	6961 Weissnat Drive Donnellyf...	6	2	Mumbai	2
Cupiditate	8	117 Nellie Summit Suite 982 So...	8	1	Bangalore	1

Cross join

The CROSS JOIN keyword returns all records from both tables (table1 and table2).
CROSS JOIN is also called Cartesian Product. It returns a result against matching Join conditions with a total of m x n rows where m and n are a number of matching rows in table1 and table2 against the JOIN condition.

MySQL CROSS JOIN syntax:

CROSS JOIN Syntax

```
SELECT column_name(s)
FROM table1
CROSS JOIN table2;
```

```
73 •    select * from Employee;
```

Result Grid | Filter Rows: | Edit: | Export/

name	id	address	Department_id	officeid
Veniam	1	640 Damon Junction East Math...	3	2
Molestiae	2	6658 Hollis Club Ernamouth, TX...	4	1
Officiis	3	59965 Mason Neck Apt. 985 Ka...	5	2
Rerum	4	91067 Geovany Fort Hanefort,...	6	1
Et	5	7647 Reva Shores Suite 970 N...	2	2
Et	6	9419 Carmela Burg Apt. 687 Ai...	8	1
Laborum	7	6961 Weissnat Drive Donnellyf...	6	2
Cupiditate	8	117 Nellie Summit Suite 982 So...	8	1

Employee

office_locations

```
74 •    select * from office_locations;
```

Result Grid | Filter Rows:

address	id
Bangalore	1
Mumbai	2
Seattle	3
Santa Clara	4

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

name	id	address	Department_id	officeid	address	id
Veniam	1	640 Damon Junction East M...	3	2	Santa Clara	4
Veniam	1	640 Damon Junction East M...	3	2	Seattle	3
Veniam	1	640 Damon Junction East M...	3	2	Mumbai	2
Veniam	1	640 Damon Junction East M...	3	2	Bangalore	1
Molestiae	2	6658 Hollis Club Ernamouth...	4	1	Santa Clara	4
Molestiae	2	6658 Hollis Club Ernamouth...	4	1	Seattle	3
Molestiae	2	6658 Hollis Club Ernamouth...	4	1	Mumbai	2
Molestiae	2	6658 Hollis Club Ernamouth...	4	1	Bangalore	1
Officiis	3	59965 Mason Neck Apt. 98...	5	2	Santa Clara	4
Officiis	3	59965 Mason Neck Apt. 98...	5	2	Seattle	3
Officiis	3	59965 Mason Neck Apt. 98...	5	2	Mumbai	2
Officiis	3	59965 Mason Neck Apt. 98...	5	2	Bangalore	1
Rerum	4	91067 Geovany Fort Hanef...	6	1	Santa Clara	4
Rerum	4	91067 Geovany Fort Hanef...	6	1	Seattle	3
Rerum	4	91067 Geovany Fort Hanef...	6	1	Mumbai	2
Rerum	4	91067 Geovany Fort Hanef...	6	1	Bangalore	1
Et	5	7647 Reva Shores Suite 97...	2	2	Santa Clara	4
Et	5	7647 Reva Shores Suite 97...	2	2	Seattle	3
Et	5	7647 Reva Shores Suite 97...	2	2	Mumbai	2
Et	5	7647 Reva Shores Suite 97...	2	2	Bangalore	1
Et	6	9419 Carmela Burg Apt. 68...	8	1	Santa Clara	4
Et	6	9419 Carmela Burg Apt. 68...	8	1	Seattle	3
Et	6	9419 Carmela Burg Apt. 68...	8	1	Mumbai	2
Et	6	9419 Carmela Burg Apt. 68...	8	1	Bangalore	1

```
SELECT * from Employee
cross JOIN office_locations;
```

Laborum	7	6961 Weissnat Drive Donn...	6	2	Santa Clara	4
Laborum	7	6961 Weissnat Drive Donn...	6	2	Seattle	3
Laborum	7	6961 Weissnat Drive Donn...	6	2	Mumbai	2
Laborum	7	6961 Weissnat Drive Donn...	6	2	Bangalore	1
Cupiditate	8	117 Nellie Summit Suite 982...	8	1	Santa Clara	4
Cupiditate	8	117 Nellie Summit Suite 982...	8	1	Seattle	3
Cupiditate	8	117 Nellie Summit Suite 982...	8	1	Mumbai	2
Cupiditate	8	117 Nellie Summit Suite 982...	8	1	Bangalore	1

What is the output?

```
SELECT * from Employee CROSS JOIN office_locations
ON
Employee.officeid = office_locations.id;
```

The output of the query:

```
83 •    SELECT * from Employee
84      cross JOIN office_locations
85      ON Employee.officeid = office_locations.id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

name	id	address	Department_id	officeid	address	id
Veniam	1	640 Damon Junction East Math...	3	2	Mumbai	2
Molestiae	2	6658 Hollis Club Ernamouth, TX...	4	1	Bangalore	1
Officiis	3	59965 Mason Neck Apt. 985 Ka...	5	2	Mumbai	2
Rerum	4	91067 Geovany Fort Hanefort,...	6	1	Bangalore	1
Et	5	7647 Reva Shores Suite 970 N...	2	2	Mumbai	2
Et	6	9419 Carmela Burg Apt. 687 Ai...	8	1	Bangalore	1
Laborum	7	6961 Weissnat Drive Donnellyf...	6	2	Mumbai	2
Cupiditate	8	117 Nellie Summit Suite 982 So...	8	1	Bangalore	1

SQL Queries with AND, OR, and NOT Logical Operators

AND, OR, and NOT are commonly used T-SQL Logical Operators that are used with a WHERE or HAVING clause to filter on more than one condition.

- **AND** displays records where ALL the conditions specified are true
- **OR** displays records where ANY of the conditions specified are true
- **NOT** displays records where the condition(s) specified are NOT TRUE

SQL AND Operator

The WHERE clause can be combined with AND, OR, and NOT operators

The AND and OR operators are used to filter records based on more than one condition:

- The AND operator displays a record if all the conditions separated by AND are TRUE.
- The OR operator displays a record if any of the conditions separated by OR is TRUE.

The NOT operator displays a record if the condition(s) is NOT TRUE.

AND Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 AND condition2 AND
condition3
...;
```

OR Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 OR condition2 OR condition3
...;
```

NOT Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE NOT condition;
```

Examples:

SELECT * FROM Customers
WHERE Country='Germany' **AND** City='Berlin';

```
SELECT * FROM Customers
WHERE City='Berlin' OR City='München';
```

```
SELECT * FROM Customers
WHERE NOT Country='Germany';
```

Addition (+) operation in SQL :

It is used to perform **addition operation** on the data items, items include either single column or multiple columns.

Implementation:

```
SELECT employee_id, employee_name, salary,
salary + 100 AS "salary + 100" FROM
Employee;
```

OUTPUT:

employee_id	employee_name	salary	salary+100
1	alex	25000	25100
2	rr	55000	55100
3	jpm	52000	52100
4	ggshmr	12312	12412

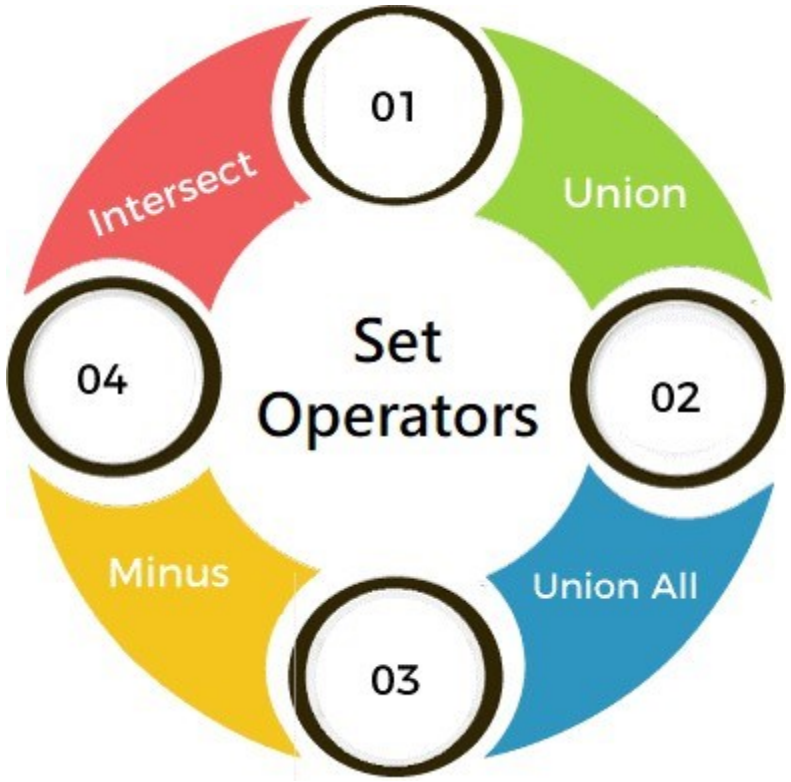
**Employee is the table name

SET Operators in SQL

SET operators are special type of operators which are used to *combine the result of two queries.*

Operators covered under SET operators are:

1. **UNION**
2. **UNION ALL**
3. **INTERSECT**
4. **MINUS**



There are certain rules which must be followed to perform operations using SET operators in SQL. Rules are as follows:

1.The number and order of columns must be the same.

2.Data types must be compatible.

Let us see each of the SET operators in more detail with the help of examples.

Consider we have the following tables with the given data.

Table 1: t_employees

ID	Name	Department	Salary	Year_of_Experience
1	Aakash Singh	Development	72000	2
2	Abhishek Pawar	Production	45000	1
3	Pranav Deshmukh	HR	59900	3
4	Shubham Mahale	Accounts	57000	2
5	Sunil Kulkarni	Development	87000	3
6	Bhushan Wagh	R&D	75000	2
7	Paras Jaiswal	Marketing	32000	1

Table 2: t2_employees

ID	Name	Department	Salary	Year_of_Experience
1	Prashant Wagh	R&D	49000	1
2	Abhishek Pawar	Production	45000	1
3	Gautam Jain	Development	56000	4
4	Shubham Mahale	Accounts	57000	2
5	Rahul Thakur	Production	76000	4
6	Bhushan Wagh	R&D	75000	2
7	Anand Singh	Marketing	28000	1

Table 3: t_students

ID	Name	Hometown	Percentage	Favourite_Subject
1	Soniya Jain	Udaipur	89	Physics
2	Harshada Sharma	Kanpur	92	Chemistry
3	Anuja Rajput	Jaipur	78	History
4	Pranali Singh	Nashik	88	Geography
5	Renuka Deshmukh	Panipat	90	Biology
6	Swati Kumari	Faridabad	93	English
7	Prachi Jaiswal	Gurugram	96	Hindi

Table 4: t2_students

ID	Name	Hometown	Percentage	Favourite_Subject
1	Soniya Jain	Udaipur	89	Physics
2	Ishwari Dixit	Delhi	86	Hindi
3	Anuja Rajput	Jaipur	78	History
4	Pakhi Arora	Surat	70	Sanskrit
5	Renuka Deshmukh	Panipat	90	Biology
6	Jayshree Patel	Pune	91	Maths
7	Prachi Jaiswal	Gurugram	96	Hindi

1. UNION:

- UNION will be used to combine the result of two select statements.
- Duplicate rows will be eliminated from the results obtained after performing the UNION operation.

Example 1:

Write a query to perform union between the table t_employees and the table t2_employees.

```
SELECT *FROM t_employees UNION SELECT  *FROM t2_employees;
```

- Here, in a single query, we have written two SELECT queries. The first SELECT query will fetch the records from the t_employees table and perform a UNION operation with the records fetched by the second SELECT query from the t2_employees table.
- You will get the following output:

ID	Name	Department	Salary	Year_of_Experience
1	Aakash Singh	Development	72000	2
2	Abhishek Pawar	Production	45000	1
3	Pranav Deshmukh	HR	59900	3
4	Shubham Mahale	Accounts	57000	2
5	Sunil Kulkarni	Development	87000	3
6	Bhushan Wagh	R&D	75000	2
7	Paras Jaiswal	Marketing	32000	1
1	Prashant Wagh	R&D	49000	1
3	Gautam Jain	Development	56000	4
5	Rahul Thakur	Production	76000	4
7	Anand Singh	Marketing	28000	1

Since we have performed union operation between both the tables, so only the records from the first and second table are displayed except for the duplicate records.

2. UNION ALL

- This operator combines all the records from both the queries.
- Duplicate rows will be not be eliminated from the results obtained after performing the UNION ALL operation.

```
SELECT *FROM t_employees UNION ALL SELECT *FROM t2_employees;
```

ID	Name	Department	Salary	Year_of_Experience
1	Aakash Singh	Development	72000	2
2	Abhishek Pawar	Production	45000	1
3	Pranav Deshmukh	HR	59900	3
4	Shubham Mahale	Accounts	57000	2
5	Sunil Kulkarni	Development	87000	3
6	Bhushan Wagh	R&D	75000	2
7	Paras Jaiswal	Marketing	32000	1
1	Prashant Wagh	R&D	49000	1
2	Abhishek Pawar	Production	45000	1
3	Gautam Jain	Development	56000	4
4	Shubham Mahale	Accounts	57000	2
5	Rahul Thakur	Production	76000	4
6	Bhushan Wagh	R&D	75000	2
7	Anand Singh	Marketing	28000	1

3. INTERSECT:

•It is used to combine two SELECT statements, but it only returns the records which are common from both SELECT statements.

Example 1:

Write a query to perform intersect operation between the table t_employees and the table t2_employees.

```
SELECT *FROM t_employees INTERSECT SELECT *FROM t2_employees;
```

Here, in a single query, we have written two SELECT queries. The first SELECT query will fetch the records from the t_employees table and perform INTERSECT operation with the records fetched by the second SELECT query from the t2_employees table.

You will get the following output:

ID	Name	Hometown	Percentage	Favourite_Subject
2	Abhishek Pawar	Production	45000	1
4	Shubham Mahale	Accounts	57000	2
6	Bhushan Wagh	R&D	75000	2

4.MINUS

•It displays the rows which are present in the first query but absent in the second query with no duplicates.

```
SELECT *FROM t_employees MINUS SELECT *FROM t2_employees;
```

ID	Name	Department	Salary	Year_of_Experience
1	Aakash Singh	Development	72000	2
3	Pranav Deshmukh	HR	59900	3
5	Sunil Kulkarni	Development	87000	3
7	Paras Jaiswal	Marketing	32000	1

Here, in a single query, we have written two SELECT queries. The first SELECT query will fetch the records from the t_employees table and perform MINUS operation with the records fetched by the second SELECT query from the t2_employees table.You will get the following output as shown.

SQL Server Aggregate Functions

Aggregate functions in SQL Server are used to **perform calculations on one or more values and return the result in a single value**. In SQL Server, all aggregate functions are built-in functions that avoid NULL values except for COUNT(*). We mainly use these functions with the GROUP BY and HAVING clauses of the SELECT statements in the database query languages.

Syntax:

The following are the syntax to use aggregate functions in SQL:

```
aggregate_function_
name(DISTINCT | ALL
 exp)
```

aggregate_function_name: It indicates the name of the aggregate function that we want to use.

DISTINCT | ALL: The DISTINCT modifier is used when we want to consider the distinct values in the calculation. The ALL modifiers are used when we want to calculate all values, including duplicates. If we do specify any modifier, all aggregate functions use the ALL modifier by default.

exp: It indicates the table's columns or an expression containing multiple columns with arithmetic operators.

Aggregate Function	Descriptions
COUNT()	This function counts the number of elements or rows, including NULL values in the defined set.
SUM()	This function calculates the total sum of all NON-NULL values in the given set.
AVG()	This function performs a calculation on NON-NULL values to get the average of them in a defined set.
MIN()	This function returns the minimum (lowest) value in a set.
MAX()	This function returns the maximum (highest) value in a set.

Comparison involving NULL and three valued logic in sql

SQL has various rules for dealing with NULL values. NULL is used to represent a missing value, but that it usually has one of three different interpretations namely:

1. Unknown value.
A person's date of birth is not known, so it is represented by NULL in the database.

2. Unavailable or withheld value.
A person has a home phone but does not want it to be listed, so it is withheld and represented as NULL in the database.

3. Not applicable attribute.
An attribute LastCollegeDegree would be NULL for a person who has no college degrees because it does not apply to that person

- When a NULL is involved in a comparison operation, the result is considered to be UNKNOWN (it may be TRUE or it may be FALSE).

- Hence, SQL uses a three-valued logic with values TRUE, FALSE, and UNKNOWN instead of the standard two-valued (Boolean) logic with values TRUE or FALSE.

- So We define the results (or truth values) of three-valued logical expressions when the logical connectives AND, OR, and NOT are used

SQL allows queries that check whether an attribute value is NULL.

Rather than using = or <> to compare an attribute value to NULL, SQL uses the comparison operators **IS or IS NOT**. This is because SQL considers each NULL value as being distinct from every other NULL value, so equality comparison is not appropriate.

Logical Connectives in Three-Valued Logic

(a)

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

(b)

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

(c)

NOT	
TRUE	FALSE
FALSE	TRUE
UNKNOWN	UNKNOWN

CREATE

CREATE TABLE

It creates a table in the database. You can specify the name of the table and the columns that should be in the table.

Syntax:

```
CREATE TABLE table_name
(column_1 datatype,
column_2 datatype,
column_3 datatype );
```

Example:

```
CREATE TABLE EMPLOYEE
(Name VARCHAR(20), Email VARCH
AR(100), DOB DATE);
```

ALTER TABLE

Changes the structure of a table. Here is how you would add a column to a database:

Syntax:

1. *//To add a new column in the table*

```
ALTER TABLE table_name ADD
column_name datatype;
```

2. *To modify existing column in the table:*

```
ALTER TABLE table_name MODIFY
(column_definitions....);
```

DROP:

It is used to delete both the structure and record stored in the table.

Syntax

```
DROP TABLE table_name;
```

Example

```
DROP TABLE EMPLOYEE;
```

INSERT: The INSERT statement is a SQL query. It is used to insert data into the row of a table.

Syntax:

INSERT INTO TABLE_NAME
(col1, col2, col3,.... col N)
VALUES (value1, value2, value3, ... valueN);

Or

INSERT INTO TABLE_NAME (columnlist)
VALUES
(value1, value2, value3, ... valueN);

For example:

INSERT INTO Textbook (Author, Subject) **VALUES** ("Sonoo", "DBMS");

SQL Modification Language – Select/delete/update

UPDATE: This command is used to update or modify the value of a column in the table.

Syntax:

```
UPDATE table_name
SET [column_name1= value1,...column_nameN = valueN]
[WHERE CONDITION]
```

For example:

```
UPDATE students
SET User_Name = 'Sonoo'
WHERE Student_Id = '3'
```

DELETE:

It is used to remove one or more row from a table.

Syntax:

```
DELETE FROM table_name
 [WHERE condition];
```

For example:

```
DELETE FROM Textbook
WHERE Author="Sonoo";
```

SELECT:

This is the same as the projection operation of relational algebra. It is used to select the attribute based on the condition described by WHERE clause.

For example:

1.SELECT emp_name
FROM employee
WHERE age > 20;

2. Select * from employee;

Syntax:

SELECT expressions
FROM TABLES
WHERE conditions;