MODULE - II

The Relational Data Model and Relational Database Constraints and Relational Algebra

Relational Model Concepts

1) **Domain** : A set of atomic values, where by "atomic" we mean simply that, from the point of the database, each value in domain is indivisible.

Examples :

i) SSN : String of digits of length.

ii) Name : String of characters beginning with an Upper case letter.

iii) Dept_Code : a member of set { MATH, ENGL, PHY, CHE ... }

2) **Attribute** : Each column in a table are called Attributes. These are the properties which define a relation. The domain of Attribute is denoted by dom(A).

Examples :

Student_RollNo, Name etc.

3) **Tuple** : A tuple is a mapping from attributes to values drawn from the respective domains of those attributes.

(OR)

It is a single row of table, which contains single record.

Examples :

A tuple for a PERSON entity might be

{ Name → "Roopa", Sex → Female, IQ → 130 }

Relation: A set of tuples of all the same form (i.e. having the same set of attributes) is called as relation.

Relational Schema: A relational schema represents the name of the relation with its attributes.

Example: STUDENT (Name, SSN, Address)

Relational Database: A collection of relations, each one consistent with its specified relational schema.

Table also called **Relation**

primary key — Domain

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive

Tuple OR ROW

Total no of rows is Cardinality.

COLUMN OR ATTRIBUTES

Total no of column is Degree

Characteristics Of Relations

Ordering of Tuples: Since relation is a set of tuples, hence they don't have any order associated with them. However, tuples in a relation can be logically ordered by the values of various attributes. In that case, information in a relation remains same, only the order of tuple varies.. Hence, tuple ordering in a relation is irrelevant.

Order of Value within tuple: A n-tuple is an ordered set of attribute values belongs to the domain D, so the order in which they appear in the tuple is significant. However, if a tuple is defined as a set of (<attribute>: <value>) pairs the order in which attributes appear irrelevant. Because there is no preference for attribute value over another.

Values and nulls in the tuples: Relational model is based on the assumption that each tuple in a relation contains a single value for each of its attribute. Hence, a relation doesnot allow composite and multivalued attributes. Moreover, it allows denoting the value of the attribute as null, if the value does not exist for that attribute or the value is known.

No tuples are identical in a relation: Since a relation is a set of tuples and a set doesnot have identical elements. Therefore, each tuple in a relation must be uniquely identified by its contents. Two tuples with same value for all the attributes cannot exist in a relation.

Interpretation of a Relation: Each relation can be viewed as a predicate and each tuple in that relation can be viewed as an assertion for which that predicate is satiesfied for the combination of values in it. In other words, each tuple represents a fact.

Relational Model Constraints

Constraints on databases can be categorized as

i) **Inherent model-based:** No tuples in a relation can be duplicates (because a relation is a set of tuples)

ii) **Schema-based:** can be expressed using DDL

iii) **application-based:** are specific to the "business rules" of miniworld and typically difficult (or) impossible to express and enforce within the data model.

Schema-based constraints are elaborated further:

(i) Domain Constraints: These can be violated if an attribute value is not appearing in the corresponding domain or it is not of the appropriate data type. Domain constraints specify that within each tuple, and the value of each attribute must be unique.

* This is specified as data types which include standard datatypes integers, real numbers, characters, Booleans, variable length strings etc.

Example:

```
Create DOMAIN CustomerName
 CHECK (value not NULL)
```

* The example shown demonstrates creating a domain constraint such that CustomerName is not NULL.

ii) Key Constraints: An attribute that can uniquely identify a tuple in a relation is called key of a table. The value of the attribute for different tuples in a relation has to be unique and their subsets of attributes is called as superkey of its relation.

A key is a minimal superkey i.e. if we were to remove any of its attributes, the resulting set of attributes fails to be superkey.

Example: Suppose that we stipulate that a faculty member is uniquely identified by NAME and address and also by NAME and department, but by no single one of the three attributes mentioned. Then {NAME, address, department} is a (non-minimal) superkey and each of {NAME, address} and {NAME, department} is a key (i.e. minimal superkey).

Candidate Key : It is any key (Hence, it is not clear what distinguishes a key from a candidate key)

Primary Key : a key chosen to act as the means by which to identify tuples in a relation. Typically, one prefers a primary key to be one having as few attributes as possible.

Entity Integrity Constraint : In a tuple, none of the values of the attributes forming the relation's primary key may have the (non-) value null. Or is it that at least one such attribute must have a non-null value.

Referential Integrity Constraint : It is based on concept of foriegn keys. A foriegn key is an important attribute of relation which should be referred to in other relationships. Referential Integrity Constraint state happens where relation refers to a key attribute of a different or same relation. However, that key element must exist in the table.

Example :

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive

Customers

Billing

Invoice No	CustomerID	Amount
1	1	1000
2	1	2000
3	2	5000

In the above example, we have two relations, Customer and Billing.

Tuple for CustomerID = 1 is referenced twice in the relation Billing. So we know CustomerName = Google has billing amount 3000.

Operations in Relational Model

Four basic operations performed on relational database model are

i) Insert is used to insert data into the relation.

ii) Delete is used to delete tuples from the table.

iii) Modify allows you to change the values of some attributes in existing tuples.

iv) Select allows you to choose a specific range of data.

Insert:

(i) domain constraint voilation: some attribute value is not of correct domain.

(ii) entity integrity voilation: key of new tuple is null.

(iii) key constraint voilation: key of new tuple is same as existing one.

(iv) referential constraint voilation: foriegn key of new tuple refers to non-existent tuple.

* The way of dealing it is reject the attempt to insert (OR) give user oppurtunity to try again with different attribute values.

Example:

The insert operation gives values of the attributes for a new tuple which should be inserted into a relation.

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive

INSERT

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

Delete:

referential integrity violation: a tuple referring to the deleted one exists.

Three options for dealing with it:

i) Reject the deletion

ii) Attempt to cascade by deleting any referencing tuples.

iii) modify the foriegn key attribute values in referencing tuples to null or to some valid value referencing a different tuple.

Example:

To specify deletion, a condition on the attributes of the relation selects the tuple to be deleted.

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

DELETE

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
4	Alibaba	Active

In the above-given example, CustomerName = "Apple" is deleted from table.

The Delete operation could violate referential integrity if the tuple which is deleted is referenced by foreign keys from other tuples in some database as mentioned above.

Update:

(i) Key constraint violation : primary key is changed so as to become same as another tuple's.

(ii) referential integrity voilation:

* foreign key is changed and new one refers to nonexistent tuple.
* primary key is changed and now other tuples that had referred to this one violate the constraint.

Example:

You can see that in the below-given relation table CustomerName = 'Apple' is updated from Inactive to Active.

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

UPDATE

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

SELECT :

In the below-given example, CustomerName = "Amazon" is selected.

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
4	Alibaba	Active

CustomerID	CustomerName	Status
2	Amazon	Active

Relational Algebra

* It is widely used procedural query language.
* It collects instances of relations as input and gives occurences of relations as output.
* It uses operations to manipulate relations.
* Used to specify retrieval requests (queries).
* Query result in the form of a relation.

Basic SQL Relational Algebra Operations

Unary Relational Operations

i) SELECT (symbol : σ)
ii) PROJECT (symbol : π)
iii) RENAME (symbol : ρ)

Relational Algebra Operations From Set Theory

i) Union ($\cup$)
ii) INTERSECTION ( ).
iii) DIFFERENCE (-)
iv) CARTESIAN PRODUCT (x)

Binary Relational Operations

i) JOIN
ii) DIVISION

SELECT (σ):

* The SELECT operation is used for selecting a subset of the tuples according to a given Selection condition.
* Sigma (σ) Symbol denotes it.
* It is used as an expression to choose tuples which meet the Selection condition.
* Select operator selects tuples that satisfy a given predicate.

$\sigma_p(r)$ where σ is predicate
r stands for relation which is the
p is prepositional logic.

Examples:

1) $\sigma_{topic = "Database"}(Tutorials)$

Output: Selects tuples from tutorials where topic = 'Database'

2) $\sigma_{sales > 50000}(customers)$

Output: Selects tuples from Customers where Sales is greater than 50000.

3) $\sigma_{(DNO = 4 \text{ AND } SALARY > 25000) \text{ OR } (DNO = 5 \text{ AND } SALARY > 30000)}(EMPLOYEE)$

Output:

FName	MINIT	LName	SSN	BDATE	ADDRESS	SEX	SALARY	DNO
Franklin	T	Wong	333445556	1955-12-08	721 Vose, Houseton	M	40000	5
Jennifer		Wellace	978654123	1941-06-20	291 Barry, TX	F	35000	4
Ramesh		Narayan	448877441	1971-03-11	975 FireOk, TX	M	39000	5

PROJECTION (π)

* The projection eliminates all attributes of the input relation but those mentioned in the projection list.
* The projection method defines a relation that contains a vertical subset of Relation.
* This helps to extract the values of specified attributes to eliminates duplicate values.
* (pi) symbol is used to choose attributes from a relation. Form of relation operation : $\pi_L(R)$.
* This operator helps you to keep specific columns from a relation and discards the other columns.

EXAMPLE :

1) $\pi_{SEX, SALARY}(EMPLOYEE)$

If several male employee have salary 30000, only a single tuple <m, 30,000> is kept in the resulting relation.

2) $\pi_{LNAME, FNAME, SALARY}(EMPLOYEE)$

LNAME	FNAME	SALARY
Smith	John	30000
Wong	Franklin	40000
Zelaya	Alicia	25000
Wallace	Jennifer	43000
Narayan	Ramesh	38000
Bong	James	55000

Ex : 2

SEX	SALARY
M	30000
M	40000
F	25000
F	43000
m	38000
m	25000
m	55000

Ex : 1

Rename :

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

EXAMPLE:

1) Retrieve the names and salaries of employees who work in department 5:

$\Pi_{FNAME, LNAME, SALARY}(\sigma_{DNO=5}(EMPLOYEE))$.

Output:-

FNAME	LNAME	SALARY
John	Smith	30000
Franklyn	Wong	40000
Ramesh	Narayan	38000
Joyce	English	25000

2) Alternatively, we specify explicit intermediate relations for each step.

$DEPT5\_EMPS \leftarrow \sigma_{DNO=5}(EMPLOYEE)$

$P \leftarrow \Pi_{FNAME, LNAME, SALARY}(DEPT5\_EMPS)$

3) Attributes can optionally be renamed in the resulting LHS relation.

$DEPT5\_EMPS \leftarrow \sigma_{DNO=5}(EMPLOYEE)$

$P(FIRSTNAME, LASTNAME, SALARY) \leftarrow \Pi_{FNAME, LNAME, SALARY}(DEPT5\_EMPS)$

TEMP	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	DNO.
	John	B	Smith	125634789	1995-2-19	731 Houston, TX	M	30,000	5
	Franklyn	T	Wong	333445556	1995-03-1	638 Vosa, Houston, TX	M	40000	5
	Ramesh	K	Narayan	999778822	1967-04-2	975 Firecircle, TX	M	39000	5
	Joyce	A	English	924721123	1966-12-2	5631 Rice, Houston, TX	F	25000	5

	FIRSTNAME	LASTNAME	SALARY
	John	Smith	30000
	Franklyn	Wong	40000
	Ramesh	Narayan	39000
	Joyce	English	25000

Union operation (∪):

UNION is symbolized by ∪ symbol. It includes all tuples that are in tables A or in B. It also eliminates duplicate tuples. So, set A UNION set B would be expressed as:

* The result ← A ∪ B
* For a union operation to be valid, the following conditions must hdd-
  i) R and S must be the same number of attributes.
  ii) Attributes domains need to be compatible.
  iii) Duplicate tuples should be automatically removed.

Example:

Consider the STUDENT and INSTRUCTOR table.

STUDENT	FN	LN
	Susan	Yao
	Ramesh	Shah
	Johny	Kohler
	Barbera	Jones
	Army	Ford
	Jimmy	Wang
	Elena	Emesl

INSTRUCTOR	FName	LName
	John	Smith
	Ricardo	Browne
	Susan	Yao
	Francis	Johson
	Ramesh	Shah.

STUDENT ∪ INSTRUCTOR

FN	LN
Susan	Yao
Ramesh	Shah
Johny	Kohler
Barbera	Jones
Army	Ford
Jimmy	Wang
Elena	Emesl
John	Smith
Ricardo	Browne
Francis	Johnson

Intersection:

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

Example:

From Union operation STUDENT and INSTRUCTION TABLE. We can find the common things in STUDENT and INSTRUCTOR i.e. STUDENT ∩ INSTRUCTOR.

FN	LN
Susan	Yao
Ramesh	Shah

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.

i) The attribute name of A has to match with the attribute name in B.

ii) The two-operand relations A and B should be either compatible or union compatible.

iii) It should be defined relation consisting of the tuples that are in relation A, but not in B.

Example:-

From the STUDENT and INSTRUCTOR table mentioned in Union operation.

(i) STUDENT − INSTRUCTOR

FN	LN
Johny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Elena	Emesyl

ii) INSTRUCTOR − STUDENT

FN	LN
John	Smith
Ricardo	Browne
Francis	Johnson

CARTESIAN PRODUCT:-

* It is an operation used to merge columns from two relations.
* Generally, a cartesian product is never a meaningful operation when it performs alone. However, it becomes meaningful when it is followed by other operations.
* It is also called as cross product.

Example:-

A

Name	Age	Sex
Ram	14	m
Sona	15	F
Kim	20	m

B

Id	Course
1	DS
2	DBMS
3	CN

If A has 'n' tuples and B has 'm' tuples then A x B will have n * m tuples.

Name	Age	Sex	Id	Course
Ram	14	m	1	DS
Ram	14	m	2	DBMS
Ram	14	m	3	CN
Sona	15	F	1	DS
Sona	15	F	2	DBMS
Sona	15	F	3	CN
Kim	20	m	1	DS
Kim	20	m	2	DBMS
Kim	20	m	3	CN

JOIN OPERATIONS:

THETA JOIN:

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

* It is similar to a CARTESIAN PRODUCT followed by SELECT. The condition θ is called join condition.

$$R(A_1, A_2, \ldots A_m, B_1, B_2, \ldots B_n) \leftarrow R_1(A_1, A_2 \ldots A_m) \bowtie_\theta R_2(B_1, B_2 \ldots B_n)$$

Example:

$$A \bowtie_{A.Column2 > B.column2} (B)$$

$A \bowtie_{A.column2 > B.column2} (B)$	
column 1	column 2
1	2

EQUI JOIN:

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

* $(A_i = B_j)$ AND ... AND $(A_h = B_k)$; $1 \le i, h \le m$, $1 \le j, k \le n$

In the above condition. (EQUI OPERATION):

i) $A_i, \ldots A_h$ are called join attributes of R_1

ii) $B_j, \ldots B_k$ are called join attributes of R_2

EXAMPLE:

1) Retreive each Department's name and it's manager's name:

$$T \leftarrow DEPARTMENT \bowtie MGRSSN = SSN\ EMPLOYEE$$

$$RESULT \leftarrow \pi_{DNAME, FNAME, LNAME}(T)$$

2) A ⋈ A.column2 = B.column2 (B)

A ⋈ A.column2 = B.column2 (B)	
column 1	column2
1	1

EQUIJOIN is most difficult operations to implement efficiently using SQL in an RDBMS.

NATURAL JOIN (*):

* Natural join can only be performed if there is common attribute between the relations. The name and type of attribute must be same.
* In a NATURAL JOIN, the redundant join attributes of R_2 are eliminated from R. The equality condition is implied and no need to specify.

$$R \leftarrow R_1 * (\text{join attributes of } R_1), (\text{join attributes of } R_2)\ R_2$$

Example:

1) Retrieve each EMPLOYEE's name and the name of the DEPARTMENT he/she works for:

$$T \leftarrow \text{EMPLOYEE} * (\text{DNO}), (\text{DNUMBER})\ \text{DEPARTMENT}$$

$$\text{RESULT} \leftarrow \pi_{\text{FNAME, LNAME, DNAME}} (T)$$

* If join attributes have the same names in both relations then $R \leftarrow R_1 * R_2$

2) Retrieve each EMPLOYEE's name and the name of his/her SUPERVISOR.

$$\text{SUPERVISOR (SUPERSSN, SFN, SLN)} \leftarrow \pi_{\text{SSN, FNAME, LNAME}} (\text{EMPLOYEE})$$

$$T \leftarrow \text{EMPLOYEE} * \text{SUPERVISOR}$$

$$\text{RESULT} \leftarrow \pi_{\text{FNAME, LNAME, SFN, SLN}} (T)$$

For database applications, additional operations are needed that were not part of original relational algebra. These include:

1) Aggregate functions and grouping.
2) OUTERJOIN and OUTER UNION.

AGGREGATE FUNCTIONS ($\mathcal{F}$):

Functions such as SUM, COUNT, AVERAGE, MIN, MAX are often applied to sets of values or set of tuples in database applications.

* $<$grouping attributes$>$ $\mathcal{F}$ $<$function list$>$ (R)

  The grouping attributes are optional

Example:

1) Retrieve the average salary of all employees

$$P(AVGSAL) \leftarrow \mathcal{F}_{AVERAGE\ SALARY}(EMPLOYEE)$$

2) For each department, retrieve the department numbers, the number of employees, and the average salary (in the dept):

$$P(DNO, NUMEMPS, AVGSAL) \leftarrow DNO\ \mathcal{F}_{COUNT\ SSN,\ AVERAGE\ SALARY}(EMPLOYEE)$$

	DNO	NUMEMPS	AVGSAL
	5	4	33250
	4	3	31000
	1	1	55000

OUTER JOIN :

* In a regular EQUIJOIN or NATURAL JOIN operation, tuples in R_1 or R_2 do not have matching tuples in other relation do not appear in the result.
* Some queries require all tuples in R_1 (or R_2 or both) to appear in the result.
* When no matching tuples are found, nulls are placed for the missing attributes.

LEFT OUTER JOIN :

* In the left outer join, operation allows keeping all tuple in left relation. However, if there is no matching tuple is found in right relation, then the attributes of right relation in the joint result are filled with null values.
* $R_1 \times R_2$ lets every tuple in R_1 appear in the result.

All rows from left table

Example :

A	
Num	Square
2	4
3	9
4	16

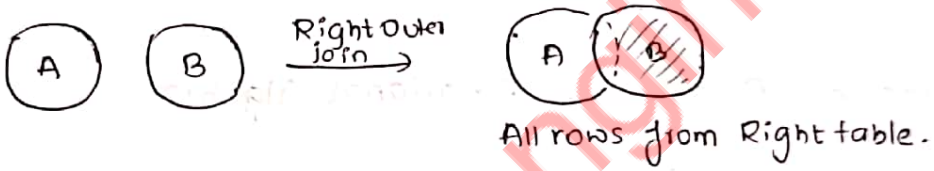
B	
Num	Cube
2	8
3	18
5	75

A ⟗ B

	A ⟗ B	
Num	Square	Cube
2	4	4
3	9	9
4	16	–

RIGHT OUTER JOIN: (A ⟖ B):

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



All rows from Right table.

Example:

From the A and B tables.

A ⟖ B

A ⟖ B		
Num	~~Squa~~ Cube	Square
2	8	4
3	18	9
5	75	–

Full outer join (A ⟗ B):

In a full outer join, all tuples from the both relations in the result, irrespective of the matching condition.

Example:

A ⟗ B

A ⟗ B		
Num	Cube	Square
2	4	8
3	9	18
4	16	-
5	-	75

Examples of Queries in Relational Algebra

1) Retrieve the name and address of all employees who work on the "Research" department.

RESEARCH_DEPT ← $\sigma_{DNAME = 'Research'}$ (DEPARTMENT)

RESEARCH_EMPS ← (RESEARCH_DEPT $\bowtie_{DNumber = DnoEMPLOYEE}$)

RESULT ← $\pi_{Fname, Lname, Address}$ (RESEARCH_EMPS)

2) For every project located in 'Stafford', list the project number, the controlling department number, and then join the result with department manager's lastname, address and birth date.

STAFFORD_PROJS ← $\sigma_{Plocation = 'Stafford'}$ (PROJECT)

CONTR_DEPTS ← (STAFFORD_PROJS $\bowtie_{Dnum = Dnumber}$ DEPARTMENT)

PROJ_DEPT_MGRS ← (CONTR_DEPTS $\bowtie_{mgr\_ssn = SSn}$ EMPLOYEE)

RESULT ← $\pi_{Pnumber, Dnum, Lname, Address, Bdate}$ (PROJ, DEPT, MGRS)

i) Find the names of employees who work on all the projects controlled by department number 5.

DEPT5_PROJS $\leftarrow \rho(Pno)\,(\pi_{Pnumber}\,(\sigma_{Dnum = 5}(PROJECT)))$

EMP_PROJ $\leftarrow \rho(ssn, Pno)\,(\pi_{Essn, Pno}(WORKS\_ON))$

RESULT_EMP_SSNS $\leftarrow$ EMP_PROJ $\div$ DEPT5_PROJS

RESULT $\leftarrow \pi_{Lname, Fname}$ (RESULT_EMP_SSNS * EMPLOYEE)

j) List the names of all employees with two or more dependents.

* This query cannot be done in the basic relational algebra. We have to use AGGREGATE FUNCTION operation with COUNT aggregate function. We assume that dependants of the same employee have distinct Dependent_name values.

$T_1(Ssn, No\_of\_dependents) \leftarrow {}_{Essn}\mathcal{F}_{COUNT\ Dependent\_name}(DEPENDENT)$

$T_2 \leftarrow \sigma_{No\_of\_dependents > 2}(T_1)$

RESULT $\leftarrow \pi_{Lname, Fname}(T_2 * EMPLOYEE)$

k) Retrieve the names of employees who have no dependents

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

ALL_EMPS $\leftarrow \pi_{Ssn}$ (EMPLOYEE)

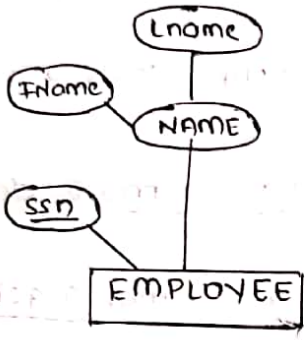
EMPS_WITH_DEPS(ssn) $\leftarrow \pi_{ssn}$ (DEPENDENT)

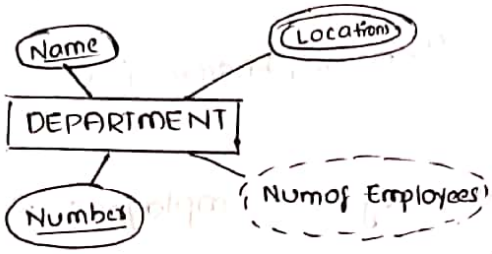
EMPS_WITHOUT_DEPS $\leftarrow$ (ALL_EMPS_WITH_DEPS)

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

Relational Database Design Using ER-to-Relational Mapping

Step 1: For each regular (strong) entity type E in the ER schema, create a relation R that includes all the simple attributes of E.





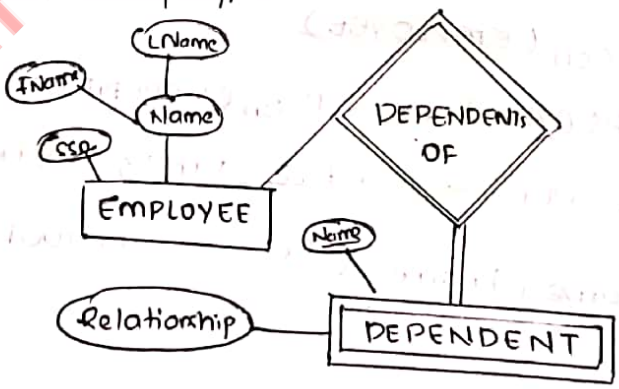
EMPLOYEE

SSN	LName	Fname

DEPARTMENT

NumBeR	NAME

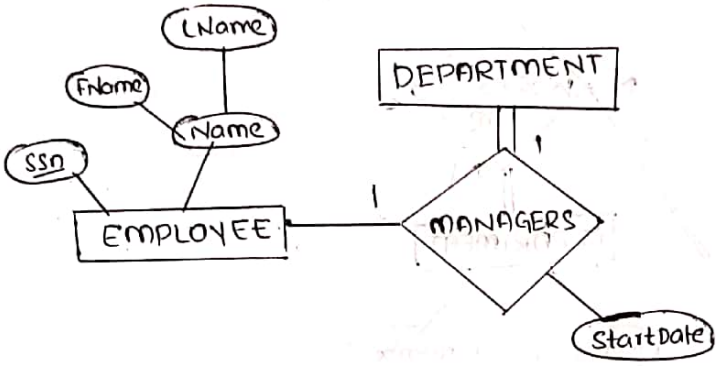
Step 2: For each weak entity type W in the ER schema with owner entity type E, create a relation R, and include all simple attributes of W as attributes. In addition, include as foreign key attributes of R the primary key attribute(s) of the relations that correspond to the owner entity types.



DEPENDENT

EMPL-SSN	NAME	Relationship

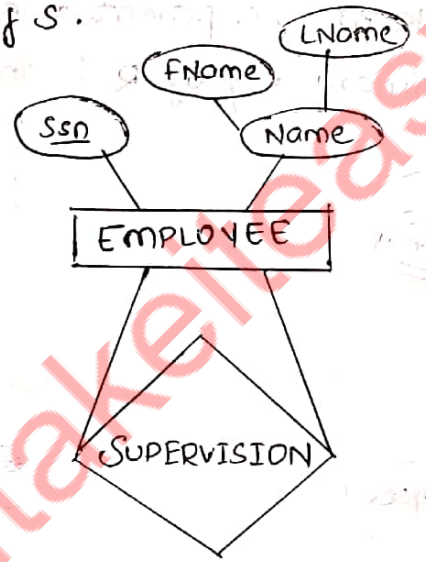
STEP 3 : For each binary 1:1 relationship type R in the ER schema, identify the relations S and T that correspond to the entity types participating in R. Choose one of the relations, say S, and include the primary key of T as a foreign key in S. Include all the simple attributes of R as attributes of S.



DEPARTMENT

MANAGER-SSN	StartDate

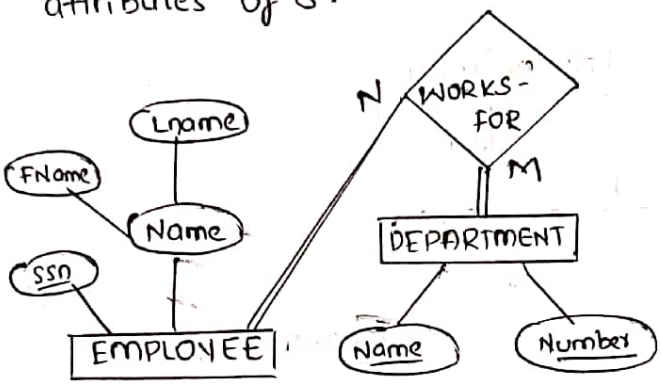
STEP 4 : For each regular binary 1:N relationship type R identify the relation (N) relation S. Include primary key of T as a foriegn key of S. Simple attributes of R map to attributes of S.



EMPLOYEE

Supervision SSN

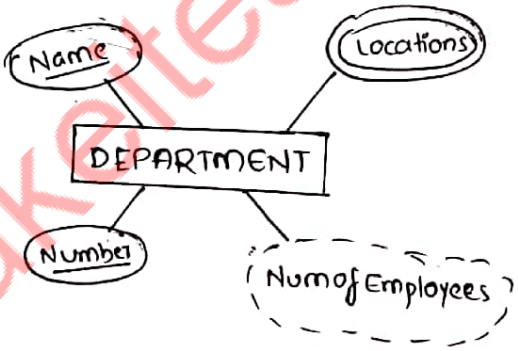
Step 5 : For each binary m:N relationship type R, create a relation S. Including the primary keys of participant relations as foriegn keys in S. Their combination will be primary key for S. Simple attributes of R become attributes of S.



WORKS-FOR

Employee SSN	DeptNumber

Step 6 : For each multi-valued attribute A, create a new relation R. This relation will include an attribute corresponding to A, plus the primary key K of the parent relation as a foreign key in R. The primary key of R is the combination of A and K.



DEP_LOCATION

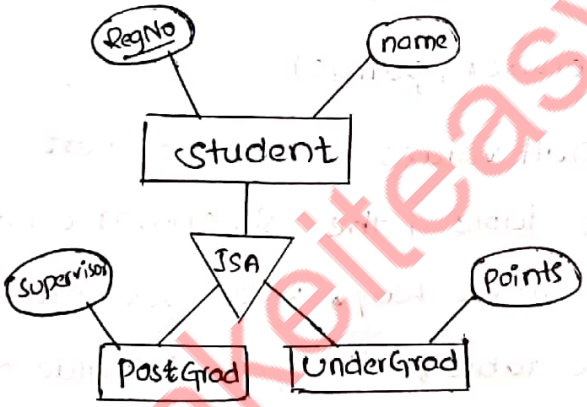
Location	DEP_NUMBER

Step 7 : For each n-ary relationship type R where n > 2, create a new relation S to represent R. Include the primary keys of relations participating in R as foreign keys in S. Simple attributes of R map to attributes of S. The primary key of S is a combination of all the foreign keys that reference the participants that have cardinality constraint > 1.

For a recursive relationship, we will need a new relation.

Mapping Specialization / generalization to relational tables

Specialization / generalization relationship can be mapped to relational tables in three methods. To demonstrate the methods, we will take the Student, pastgraduate and undergraduate relationship. A student in the university has a registration number and a name. Only postgradluate students have supervisors. undergraduates accumalates points through their coursework.



Method 1 :

All the entities in the relationship are mapped to individual tables.

Student (Regno, name)

PosGrad (Regno, supervisor)

UnderGrad (Regno, points)

Method 2:

* Only Subclasses are mapped to tables. The attributes in the Superclass are duplicated in all subclasses.

  PosGrad (Regno, name, Supervisor)

  UnderGrad (Regno, name, points)

* This method is most preferred when inheritance is disjoint and complete.

  Example:

  - Every student is either PosGrad or UnderGrad and nobody is both.

Method 3:

* Only the Superclasses is mapped to a table. The attributes in the Subclasses are taken to the Superclass.

  Student (Regno, name, supervisor, points)

* This method will introduce null values. When we insert an undergraduate record in the table, the supervisor column value will be null. In the same way, when we insert a postgraduate record in the table, the points value will be null.

SQL

The SQL language may be considered one of the major reasons for the commercial success of relational databases.

* It has statements for data definations, queries, and updates. Hence, it is both a DDL and a DML. It also has rules for embedding SQL statements into a general purpose programming language such as Java, or C/C++.

SQL Data Defination and Data types :-

* SQL uses the terms table, row, and column for the formal relational terms relation tuple, and attribute respectively.
* The main SQL Command for data defination is the CREATE statement, which can be used to create schemas, tables, and domains.

Schema and Catalog concepts in SQL

* An SQL schema identified by a schema name, and includes an authorization identifier to indicate the user or account who owns schema, as well as descriptors for each element in schema.
* Schema elements include tables, constraints, views, domains and other constructs that describe the schema.
* A schema is created via the CREATE SCHEMA statement, which can include all the schema elements definations.
* The schema can be assigned a name and authorization identifier, and the elements can be defined later.

* For example, the following statement creates a schema called COMPANY, owned by the user with authorization identifier 'Jsmith'. Note that SQL statement ends with semicolon

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

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

CREATE TABLE Command in SQL :-

* 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, entitity 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.
* We can explicitly attach schema name to table name. For example,

```
CREATE TABLE COMPANY.EMPLOYEE
```

* The relations declared through CREATE TABLE statements are called base tables; this means that the relation and its tuples are actually created and stored as a file by the DBMS.

Example:

```
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));
```

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). Formatted numbers can be declared by using DECIMAL (i,j) - or DEC(i,j) or NUMBER (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.
* Character-string data types are either fixed length - CHAR(n), Where n is the number of characteristics, VARCHR(n), where n is the maximum number of characters.
  * When specifying a literal string value, it is placed between single quotation marks and it is case sensitive.
  * For fixedlength strings, a shorter string is padded with blank characters to the right.
  * For example, if the value 'Smith' is for an attribute of type CHAR(10), it is padded with five blank characters to become 'Smith ' if needed.
* Bit-string data types are either of fixed length n - BIT(n) - Where n is the maximum number of bits. The default for n, the length of a character string is 1. Literal bit strings are placed between single quotes but preceded by a B to distinguish them from character strings.

for example, B'10101'

* 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 atleast 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 fraction of seconds and an optional WITH TIME ZONE qualifier. Literal values are represented by singlequoted strings preceded by the keyword TIMESTAMP, with a blankspace between data and time;

for example, TIMESTAMP '2008-09-27 09:12:47.648302'

* Another datatype related to DATE, TIME and TIMESTAMP is the INTERVAL data type. This specifies an interval - a relative value that can be used to increment or decrement an absolute value of a date, time, or timestamp. Intervals are qualified either YEAR/MONTH intervals or DAY/TIME intervals.

Specifying Constraints in SQL

The basic constraints that can be specified in SQL as part of table creation. These 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 attributes 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.
* It is possible to define a default value for an attribute by appending the clause DEFAULT <value> to an attribute or domain defination.
* Another type of constraint can restrict attribute or domain values using the CHECK clause following an attribute or domain defination.
* For example, suppose that department numbers are restricted to integer numbers between 1 and 20; then we can change the attribute declaration of Dnumber in the DEPARTMENT table to the following:

```
Dnumber INT NOT NULL CHECK ( Dnumber > 0 AND Dnumber < 21);
```

* The CHECK clause can be used in conjunction with CREATE DOMAIN.

```
CREATE DOMAIN D_NUM1 AS INTEGER CHECK ( D_NUM1 > 0 AND D_NUM1 < 21);
```

Ex:

```
CREATE TABLE EMPLOYEE ( Dno INT NOT NULL DEFAULT 1 , CONSTRAINT
EMPPK PRIMARY KEY ( Ssn ) , CONSTRAINT EMPDEPTFK FOREIGN KEY (Dno)
REFERENCES DEPARTMENT ( Dnumber ) ON DELETE SET DEFAULT ON
UPDATE CASCADE );
```

Specifying Key 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.

  EX : Dnumber INT PRIMARY KEY;

* The UNIQUE clause specifies alternate (secondary) keys, as illustrated in the DEPARTMENT and PROJECT table declarations.

  EX : Dname VARCHAR(15) UNIQUE;

* Referential integrity is specified via the FOREIGN KEY clause. A referential integrity constraint can be violated when tuples are inserted or deleted, or when FK or PK attribute value is modified.

* The default action that SQL takes for an integrity violation is to reject the update operation that will cause violation, which is known as the RESTRICT option.

* However, the schema designer can specify alternative action i.e. referential triggered action clause is attached to any FK. The options include SETNULL, CASCADE, and SET DEFAULT. An option must be qualified with either ON DELETE or ON UPDATE.

* Here, the database designer chooses ON DELETE SET NULL and ON UPDATE CASCADE for the foreign key Super_ssn of EMPLOYEE.

* This means if tuple for a supervising employee is deleted, the value of Super_ssn is automatically set to NULL for all the employee tables that were referencing the deleted employee tuple.

* If the Ssn value for a supervising employee is updated, the new value is cascaded to super_ssn for all employee tuples referencing the updated employee tuple.

Giving Names to Constraints

* A constraint may be given a constraint name, following the word CONSTRAINT. The names of all constraints within a particular schema must be unique.

* A constraint name is used to identify a particular constraint in case the constraint must be dropped later and replaced with another constraint. Giving names to constraints is optional.

EX: CONSTRAINT EMPK PRIMARY KEY (ssn).

Specifying Constraints on Tuple Using CHECK

* In addition to key and referential integrity constraints, which are specified by special keywords, other table constraints can be specified through additional CHECK clauses at the end of a CREATE TABLE statement.

* These can be called tuple-base constraints because they apply to each tuple individually and are checked whenever a tuple is inserted or modified.

* Example:

CHECK (Dept_create_date <= mgr_start_date);

In DEPARTMENT table, we can add additional attribute Dept_create_date, which stores the date when the department was created. Then we could add the CHECK clause at the end of CREATE TABLE statement for the DEPARTMENT table to make sure that a manager's start date is later than department creation date.

Retrieval Queries in SQL

SQL allows a table to have two or more tuples that are identical in all their attribute values. Hence, in general, an SQL table is not a set of tuples, because a set does not allow two identical members; rather, it is a multiset of tuples.

The SELECT-FROM-WHERE Structure of Basic SQL Queries

* The basic form of SELECT statement, sometimes called a mapping or a select-from-where block, is formed of the three clauses SELECT, FROM, and WHERE and has following form:

```
SELECT <attribute list>
FROM <table list>
WHERE <condition>
```

where

* <attribute list> is a list of attribute names whose values are to be retrieved by the query.
* <table list> is a list of the relation names required to process the query.
* <condition> is a conditional (Boolean) expression that identifies the tuples to be retrieved by the query.

Query 1

1) Retrieve the birthdate and address of the employees whose name is 'John B. Smith'.

```
SELECT Bdate, Address FROM EMPLOYEE WHERE Fname='John' AND Minit='B'
AND Lname='Smith';
```

Bdate	Address
1965-01-09	731 Fondren, Houston TX

2) Retrieve the name and address of all employee who work for the 'Research' department.

SELECT Fname, Lname, Address FROM EMPLOYEE, DEPARTMENT WHERE Dname='Research' AND Dnumber = Dno.

Fname	Lname	Address
John	Smith	731 Fondren, Houston, TX
Franklyn	Wong	638 Voss, Houston, TX
Ramesh	Narayan	975 FireOak, Humble, TX
Joyce	English	5631 Rice, Houston, TX

* A query that involves only selection and join conditions plus projection attributes is known as select-project-join query. The next example is a select-project-join query with two join conditions.

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.

SELECT Pnumber, Dnum, Lname, Address, Bdate FROM PROJECT, DEPARTMENT EMPLOYEE WHERE Dnum = Dnumber AND mgr_ssn = Ssn AND Plocation = 'Stafford';

Pnumber	Dnum	Lname	Address	Bdate
10	4	Wallace	291 Berry, Bellaire, TX	1941-06-20
30	4	Wallace	291 Berry, Bellaire, TX	1941-06-20

The join condition Dnum = Dnumber relates a project tuple to its controlling department tuple, whereas the join condition mgr_ssn = Ssn relates the controlling department tuple to the employee tuple who manages that department.

Ambiguous Attribute Names, Aliasing, Renaming, and Tuple Variables

* In SQL, the same name can be used for two (or more) attributes as long as the attributes are in different relations. If this is the case, and a multitable query refers to two or more attributes with the same name, we must qualify the attribute name with the relation name to prevent ambiguity.

* This is done by prefixing the relation name to the attribute name and seperating the two by a period.

```
SELECT Fname, EMPLOYEE.Name, Address FROM EMPLOYEE, DEPARTMENT
WHERE DEPARTMENT.Name = 'Research' AND DEPARTMENT.Dnumber =
EMPLOYEE.Dnumber;
```

* We can also create an alias for each table name to avoid repeated typing of long table names.

* For each employee, retrieve the employee's first and last name and the first and last name of his or her immediate ~~spon~~ supervisor.

```
SELECT E.Fname, E.Lname, S.Fname, S.Lname
FROM EMPLOYEE AS E, EMPLOYEE AS S WHERE E.Super_ssn = S.Ssn;
```

EFname	Elname	S.Fname	S.Lname
John	Smith	Franklyn	Wong
Franklyn	Wong	James	Bong
Alicia	Zelaya	Jennifer	Wallace
Jennifer	Wallace	James	Bong
Ramesh	Narayan	Franklyn	Wong
Joyce	English	Franklyn	Wong
Ahmad	Jabbar	Jennifer	Wallace

* In this case, we are required to declare alternative relation names E and S, called aliases or tuple variable, for EMPLOYEE relation. An alias can follow the keyword AS, as shown in Query, or it can directly follow the relation name

* It is also possible to rename the relation attributes within the query in SQL by giving them aliases.

For example, if we write EMPLOYEE AS E(Fn, Mi, Ssn, Bd, Addr, Sex, Sal, Ssn, Dno) in the FROM clause, Fn become an alias for Fname, Mi for Minit, Ln for Lname. —— so on.

* In Query we can think of E and S as two different copies of the EMPLOYEE relation; the first, E, represents employees in the role of supervisees; the second, S, represents employee in the role of supervisors.

Unspecified WHERE clause and Use of the Asterisk

* A missing WHERE clause indicates no condition on tuple selection; hence, all tuples of the relation specified in the FROM clause qualify and are selected for the query result. If more than one relation is specified in the FROM clause and there is no WHERE clause, then the CROSS PRODUCT – all possible tuple combinations.

Queries 9 and 10.

Select all EMPLOYEE Ssns (Q9) and all combinations of EMPLOYEE Ssn and DEPARTMENT Dname (Q10) in the database.

Q9: SELECT Ssn FROM EMPLOYEE;

972123456
123456789
111222333
445566778
112231233

Q_{10} : SELECT Ssn, Dnome FROM EMPLOYEE, DEPARTMENT;

Ssn	Dname
972123456	Research
123456789	Research
111222333	Research
445566778	Research
112231233	Research
972123456	Administration
123456789	Administration
111222333	Administration
445566778	Administration
112231233	Administration
972123456	Headquarter
123456789	Headquarter
111222333	Headquarter
445566778	Headquarter
112231233	Headquarter

* It is important to specify every selection and join condition in the WHERE clause, if any such condition is overlooked, incorrect and very large relations may result.

Tables as Sets in SQL

* SQL usually treats a table not as a set but rather as a multiset; duplicate tuples can appear more than once in a table, and in the result of a query. SQL does not automatically eliminate duplicate tuples in the result of queries.

* If we donot eliminate duplicate tuples from the result of an SQL query, we use the keyword DISTINCT in the SELECT clause, meaning that only distinct tuples should remain in the result.

Query 11

Retrieve the Salary of every employee (Q11) and all distinct Salary values. (Q11).

Q11 : SELECT ALL Salary FROM EMPLOYEE ;

Q11A : SELECT DISTINCT Salary FROM EMPLOYEE ;

In general, a query with SELECT DISTINCT eliminates duplicates, whereas a query with SELECT ALL does not.

a)

Salary
30000
40000
25000
25000

b)

Salary
30000
40000
25000

* SQL has directly encorporated some of the set operations they are set union (UNION), Cset difference (EXCEPT) and set intersection (INTERSECT) thus the relations resulting from these operations are set of tuples; that is, duplicate tuples are eliminated from the result.

* These set operations apply only to union-compatible relations, so we must make sure that the two relations on which we apply only to union operation have the same attributes and that the attributes appear in the same order in both relations.

Query

Make a list of all 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.

(SELECT DISTINCT Pnumber FROM PROJECT, DEPARTMENT, EMPLOYEE WHERE Dnum = Dnumber AND Mgr_ssn = Ssn AND Lname = 'Smith')

UNION

(SELECT DISTINCT Pnumber FROM PROJECT, WORKS_ON, EMPLOYEE WHERE Pnumber = Pno AND Essn = Ssn AND Lname = 'Smith');

Substring Pattern Matching and Arithmetic Operators

* The first feature allows comparision conditions on only parts of a character string, using the LIKE comparision operator. This can be used for string pattern matching.
* Partial strings are specified using two reserved characters: % replaces an arbitary number of zero (or) more characters, and the underscore (_) replaces a single character.

Query:- Retrieve all employees whose address is in Houston, Texas.

```
SELECT Fname, Lname FROM EMPLOYEE WHERE ADDRESS LIKE '%Houston,TX%';
```

Query 12A : Find all employees who were born during the 1950s

```
SELECT Fname, Lname FROM EMPLOYEE WHERE Bdate LIKE '_ _ 5 _ _ _ _ _ _';
```

* Another feature allows the use of arithmetic in queries. The standard arithmetic operators for addition (+), subtraction (-), multiplication (*), and division (/) can be applied to numeric values or attributes with numeric domains.

Query 13: Show the resulting salaries if every employee working on the 'PRODUCTX' project is given a 10 percent raise.

```
SELECT E.Fname, E.Lname, 1.1 * E.Salary AS INCREASED_SAL FROM EMPLOYEE
AS E, WORKS_ON AS W, PROJECT AS P WHERE E.Ssn = W.Pno = P.number
AND P.Pname = 'PRODUCTX'.
```

* Another comparision operator, which can be used for convenience, is BETWEEN, which illustrated in Query 14.

Query 14: Retreive all employees in department 5 whose salary is between $30,000 and $40,000.

```
SELECT * FROM EMPLOYEE WHERE (Salary BETWEEN 30000 AND 40000)
AND Dno = 5;
```

The condition = ((Salary >= 30000) AND (Salary <= 40000))

Ordering of Query Results

SQL allows the user to order the tuples in the result of a query by the values of one or more of the attributes that appear in the query result, by using the ORDER BY clause.

Example: SELECT Salary FROM EMPLOYEE ORDER BY Salary;

* The default order is in ascending order of values. We can specify the keyword DESC if we want to see the result in a descending order of values. The keyword ASC can be used to specify ascending order explicitly.

Insert Command :-

* Insert is used to add a single tuple to a relation. We must specify the relation name and a list of values for the tuple. The values should be listed in the same order in which the corresponding attributes were specified in the CREATE TABLE command.

Ex:- INSERT INTO EMPLOYEE VALUES ('Richard', 'K', 'Marini', '923456123', '1962-12-30', '98 Oak Forest, Katy, Tx', 'M', 37000, 4);

* A second form of the INSERT statement allows the user to specify explicit attributes names that correspond to the values provided in the INSERT command. This is useful if a relation has many attributes but only a few of those attributes are assigned values in the new tuple.

Ex:- INSERT INTO EMPLOYEE (Fname, Lnome, Dno, Ssn) VALUES ('Richard', 'marini', 4, '923456123');

* Attributes not specified in the above query are set to their DEFAULT or to NULL, and the values are listed in the same order as the attributes are listed in the INSERT command itself.

* A variation of the INSERT command inserts multiple tuples into a relation in conjunction with creating the relation and loading it with the result of a query.

Example:

```
CREATE TABLE WORKS_ON_INFO ( Emp_name VARCHAR(15), Proj_name
VARCHAR(15), Hours_per_week DECIMAL(3,1));

INSERT INTO WORKS_ON_INFO( Emp_name, PROJ_name, Hours_per_week)
SELECT E.Lname, P.Pname, W.Hours FROM PROJECT P, WORKS_ON W,
EMPLOYEE E WHERE P.Pnumber = W.Pno AND W.Essn = E.Ssn;
```

The DELETE Command

* The DELETE command removes tuples from a relation it. It includes a WHERE clause, similar to that used in an SQL query, to select the tuples to be deleted. Tuples are explicitly deleted from only one table at a time.
* However, the deletion may propagate to tuples in other relations if referential triggered actions are specified in the referential in the constraints of the DDL.
* Depending on the number of tuples selected by the condition in the WHERE clause, zero, one, or several tuples can be deleted by a single DELETE command. A missing WHERE clause specifies that all tuples in the relation are to be deleted; however, the table remains in the database as an empty table.

```
Ex: DELETE FROM EMPLOYEE WHERE Lname = 'Brown';
    DELETE FROM EMPLOYEE WHERE Ssn = '123456789';
    DELETE FROM EMPLOYEE WHERE Dno = 5;
    DELETE FROM EMPLOYEE;
```

The UPDATE Command

* The UPDATE command is used to modify attribute values of one or more selected tuples. As in the DELETE command, a WHERE clause in the UPDATE command selects the tuples to be modified from a single relation.

* However, updating a primary key value may propagate to the foreign key values of tuples in other relations if such a referential triggered action is specified in the referential integrity constraints of the DDL. An additional SET clause in the UPDATE command specifies the attributes to be modified and their new values.

* For example, to change the location and controlling department number of project number 10 to 'Bellaire' and 5, respectively.

```
UPDATE PROJECT SET Plocation = 'Bellaire', Dnum = 5 WHERE
Pnumber = 10;
```

* Several tuples can be modified with a single UPDATE command. An example is to give all employees in the 'Research' department a 10 percent raise in salary.

```
UPDATE EMPLOYEE SET Salary = Salary * 1.1 WHERE Dno = 5;
```

* It is also possible to specify NULL or DEFAULT as the new attribute value.

Additional Features of SQL

SQL has a number of additional features. these are follows:

SQL has various techniques for specifying complex retrieval queries, including nested queiries, aggregate functions, grouping, joined tables, outer joins, and recursive queries; SQL views, triggers, and assertions; and commands for schema modification.

* SQL has various techniques for writing programs in various programming languages that include SQL statements to access one or more databases. These include embedded (and dynamic) SQL, SQL / CLI (call Level Interface) and its predecessor ODBC (open Data Base Connectivity), and SQL / PSM (persistent Stored Modules).

* Each commercial RDBMS will have, in addition to the SQL commands, a set of commands for specifying physical database design parameters, file structures for relations, and access paths such as indexes.

* SQL has transaction control commands. These are used to specify units of database processing for concurrency control and recovery purposes.

* SQL has language constructs for specifying the granting and revoking of privileges to users. Privileges typically correspond to the right to use certain SQL commands to access certain relations. Each relation is assigned an owner, and either the owner or the DBA staff can grant to selected users the privilege to use an SQL statement - such as SELECT, INSERT, DELETE or UPDATE - to access relation.

In addition, the DBA staff can grant the privileges to create schemas, table, or views to certain users. These SQL commands - called GRAND and REVOKE -

* SQL has language constructs for creating triggers. these are generally referred to as active database techniques, since they specify actions that are automatically triggered by events such as database updates.

* SQL has incorporated many features from OO models to have more powerful capabilities, leading to enhanced relational systems known as object - relational.

* SQL and relational databases can interact with new techniques such as XML.