DATABASE MANAGEMENT SYSTEMS

MODULE-2

Introduction to E-R Model

E-R Model: The ER model defines the conceptual view of a database. It works around real-world entities and the associations among them. At view level, the ER model is considered a good option for designing databases.

The main components of E-R model are: Entity, Entity Type, Entity Set, Attributes, Relationship Type and Relationship Set.

Entity: An Entity may be an object that can be used in database with a physical existence – a particular person, car, house, or employee – or it may be an object with a conceptual existence – a company, a job, or a university course.

In ER diagram, this can be represented in rectangle shape.

Entity Type: The collection of entities that have the same attributes is called entity type.

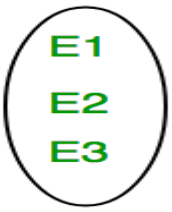
Entity Set: An entity set is a collection of similar types of entities. An entity set may contain entities with attribute sharing similar values.

Example:

If a Student is an Entity, then the complete dataset of all the students will be the Entity Set.

An Entity is an object of Entity Type and set of all entities is called as entity set. e.g.; E1 is an entity having Entity Type Student and set of all students is called Entity Set. In ER diagram, Entity Type is represented as:

Entity Type



Entity Set

- The Entity which is having its own Primary Key is called Strong Entity
- The Entity which is not having its own Primary Key is called Weak Entity

Strong Entity

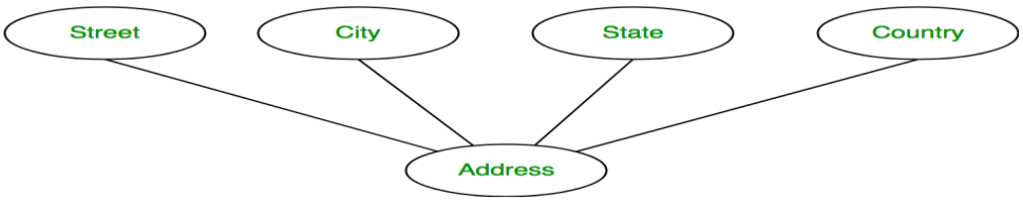
Weak Entity

Attributes: Attributes are the properties which define the entity type. For example, Roll_No, Name, DOB, Age, Address, Mobile_No are the attributes which defines entity type Student. In ER diagram, attribute is represented by an oval.

Types of attributes:

1. Key Attribute: The attribute which uniquely identifies each entity in the entity set is called key attribute. For example, Roll_No will be unique for each student. In ER diagram, key attribute is represented by an oval with underlying lines.

2. Composite Attribute: An attribute composed of many other attribute is called as composite attribute. For example, Address attribute of student Entity type consists of Street, City, State, and Country. In ER diagram, composite attribute is represented by an oval comprising of ovals.

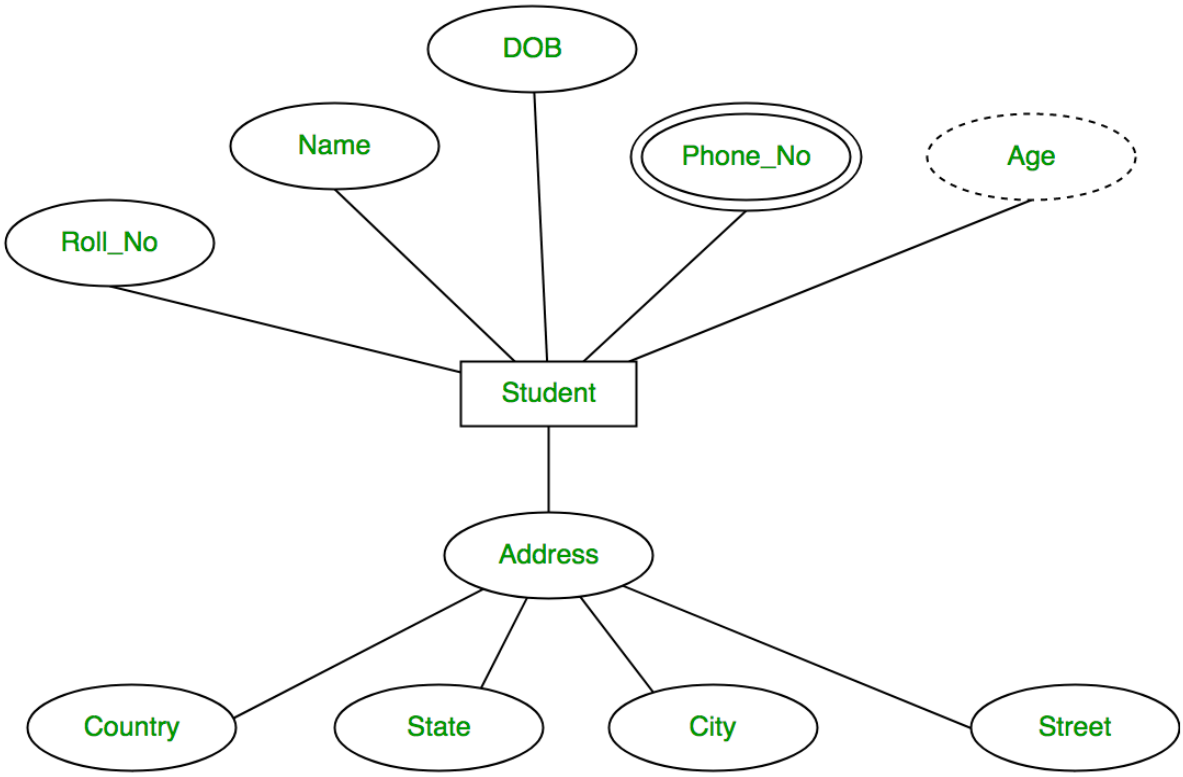


3. Multivalued Attribute: An attribute consisting more than one value for a given entity. For example, Phone_No (can be more than one for a given student). In ER diagram, multivalued attribute is represented by double oval.

4. Derived Attribute: An attribute which can be derived from other attributes of the entity type is known as derived attribute. e.g.; Age (can be derived from DOB). In ER diagram, derived attribute is represented by dashed oval.

5. Simple attribute: The attributes with values that are atomic and cannot be broken down further are simple attributes.

The complete entity type Student with its attributes can be represented as:

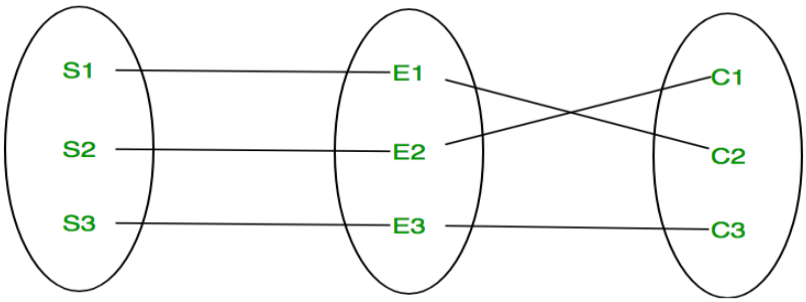


Relationships: When an Entity is related to another Entity, they are said to have a relationship. For example, A Class Entity is related to Student entity, because students study in classes, hence this is a relationship.

Relationship Type: A relationship type represents the association between entity types. For example, 'Enrolled in' is a relationship type that exists between entity type Student and Course.

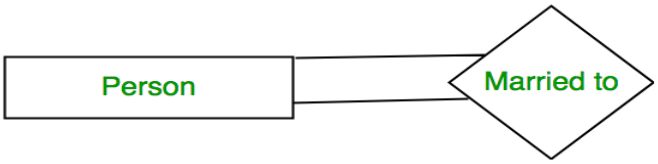
In ER diagram, relationship type is represented by a diamond and connecting the entities with lines.

Relationship Set: A set of relationships of same type is known as relationship set. The following relationship set depicts S1 is enrolled in C2, S2 is enrolled in C1 and S3 is enrolled in C3.

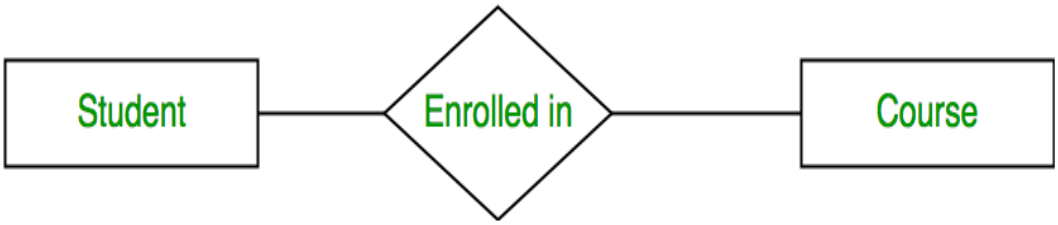


Degree of Relationship: The number of different entity sets participating in a relationship defines the degree of the relationship.

1. Unary Relationship –
   When there is only ONE entity set participating in a relation, the relationship is called as unary relationship. For example, one person is married to only one person.



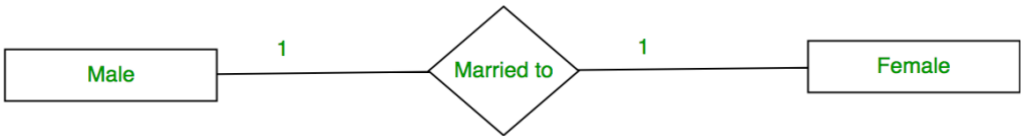
2. Binary Relationship –
   When there are TWO entities set participating in a relation, the relationship is called as binary relationship.For example, Student is enrolled in Course.



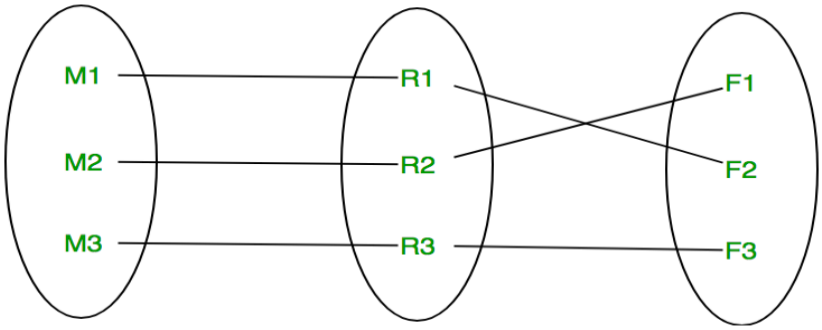
3. n-ary Relationship –
   When there are n entities set participating in a relation, the relationship is called as n-ary relationship.

Cardinality: The number of times an entity of an entity set participates in a relationship set is known as cardinality. Cardinality can be of different types:

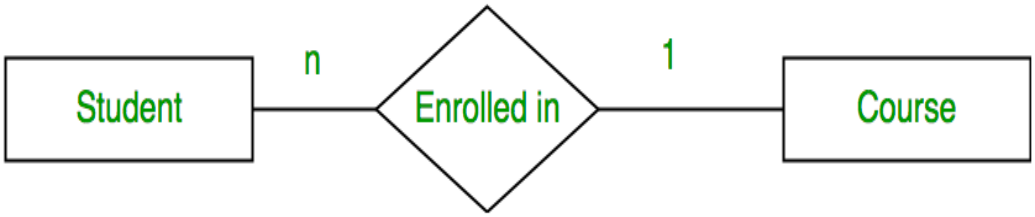
1. One to one – When each entity in each entity set can take part only once in the relationship, the cardinality is one to one. Let us assume that a male can marry to one female and a female can marry to one male. So the relationship will be one to one.



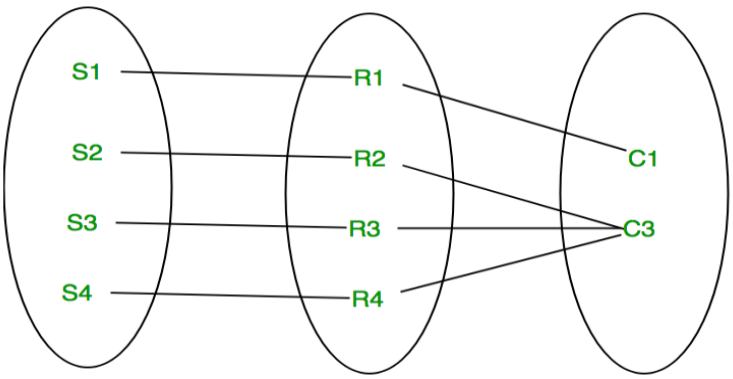
Using Sets, it can be represented as:



2. Many to one – When entities in one entity set can take part only once in the relationship set and entities in other entity set can take part more than once in the relationship set, cardinality is many to one. Let us assume that a student can take only one course but one course can be taken by many students. So the cardinality will be n to 1. It means that for one course there can be n students but for one student, there will be only one course.



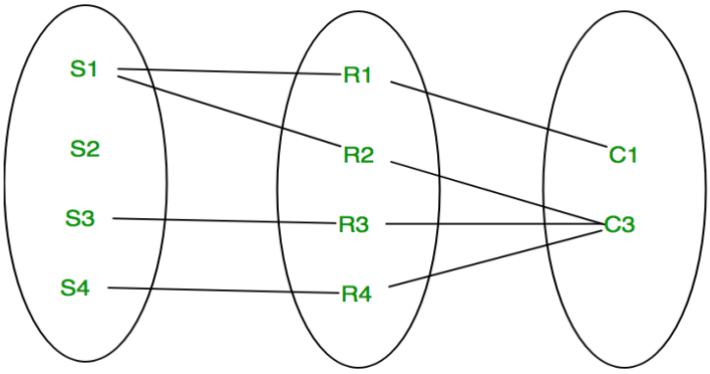
Using Sets, it can be represented as:



In this case, each student is taking only 1 course but 1 course has been taken by many students.

3. Many to many – When entities in all entity sets can take part more than once in the relationship cardinality is many to many. Let us assume that a student can take more than one course and one course can be taken by many students. So the relationship will be many to many.

Using sets, it can be represented as:

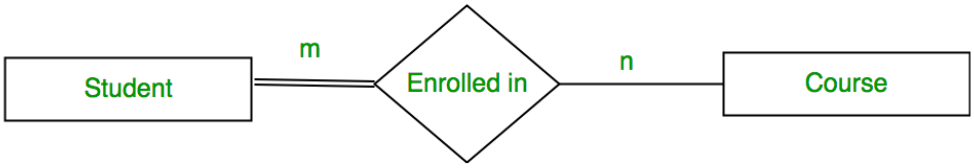


In this example, student S1 is enrolled in C1 and C3 and Course C3 is enrolled by S1, S3 and S4. So it is many to many relationships.

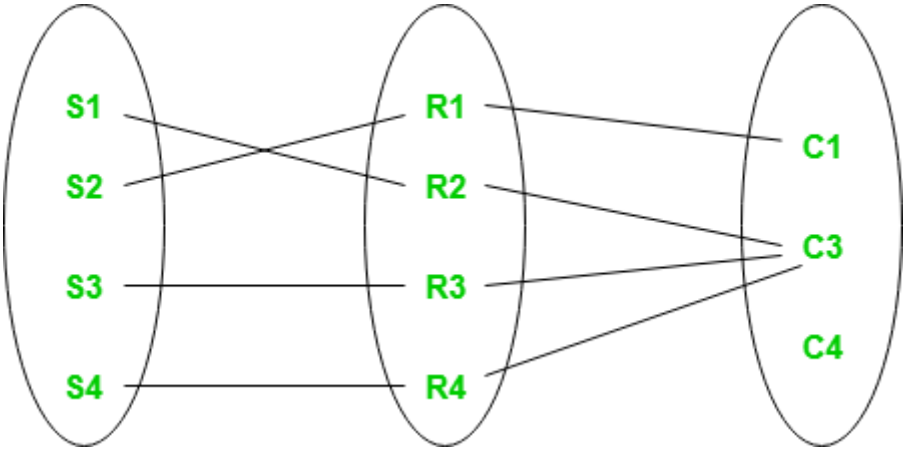
Participation Constraint: Participation Constraint is applied on the entity participating in the relationship set.

1. Total Participation – Each entity in the entity set must participate in the relationship. If each student must enrol in a course, the participation of student will be total. Total participation is shown by double line in ER diagram.
2. Partial Participation – The entity in the entity set may or may NOT participate in the relationship. If some courses are not enrolled by any of the student, the participation of course will be partial.

The diagram depicts the 'Enrolled in' relationship set with Student Entity set having total participation and Course Entity set having partial participation.



Using set, it can be represented as,

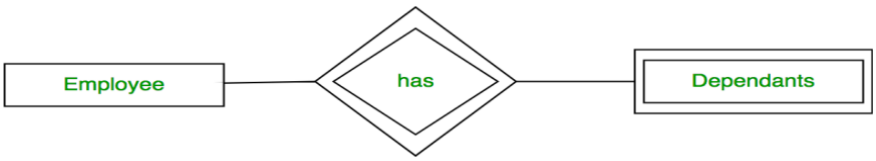


Every student in Student Entity set is participating in relationship but there exists a course C4 which is not taking part in the relationship.

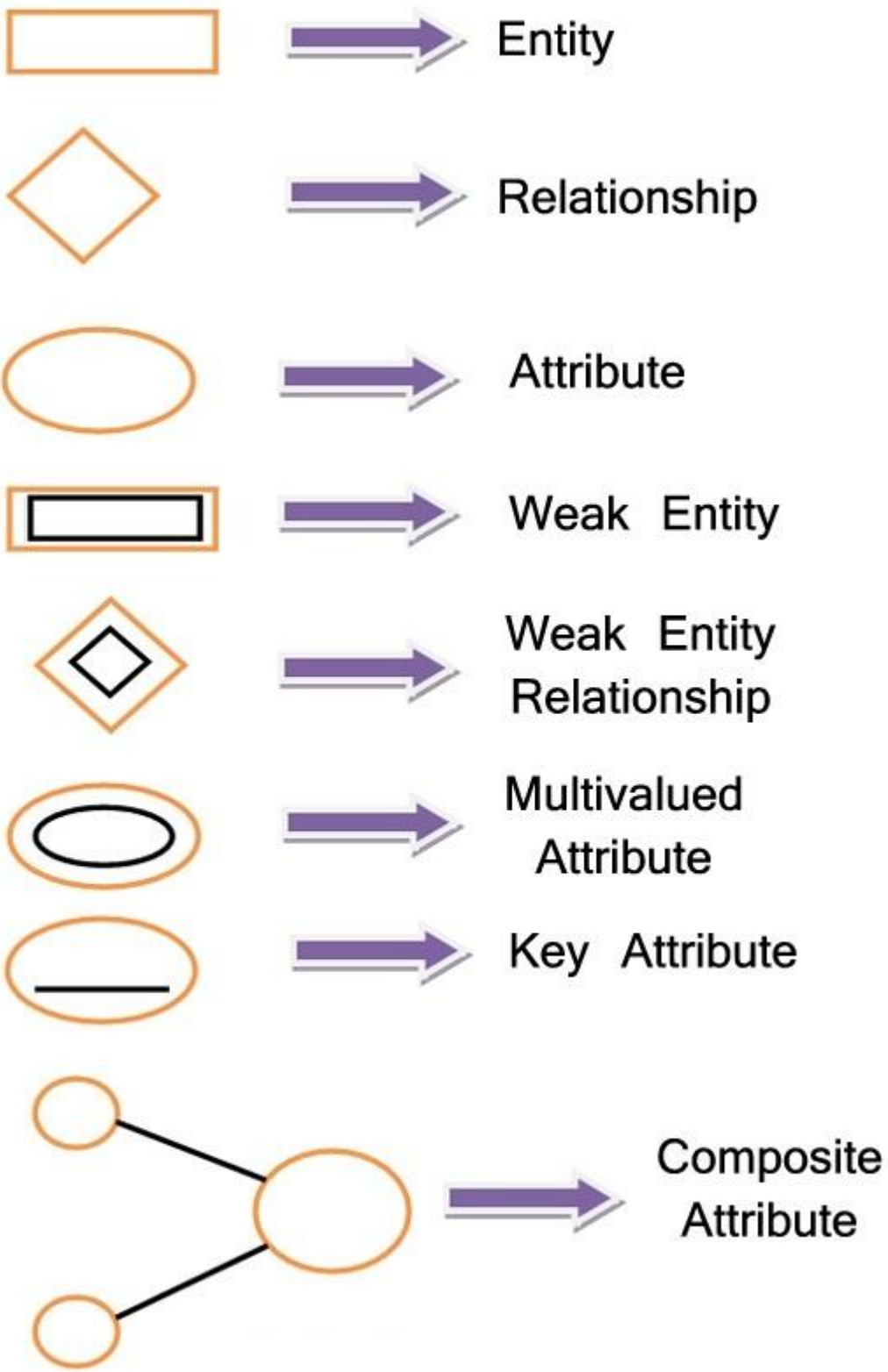
Weak Entity Type and Identifying Relationship: As discussed before, an entity type has a key attribute which uniquely identifies each entity in the entity set. But there exists some entity type for which key attribute can't be defined. These are called Weak Entity type.

For example, a company may store the information of dependants (Parents, Children, Spouse) of an Employee. But the dependents don't have existence without the employee. So Dependent will be weak entity type and Employee will be Identifying Entity type for Dependant.

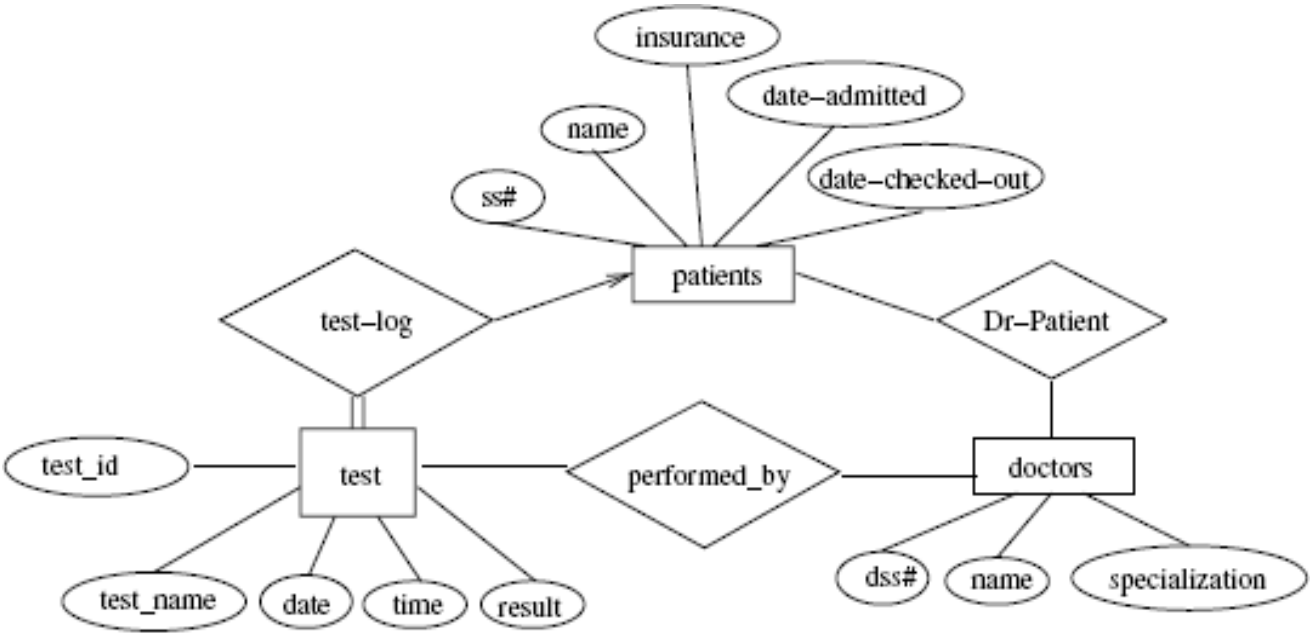
A weak entity type is represented by a double rectangle. The participation of weak entity type is always total. The relationship between weak entity type and its identifying strong entity type is called identifying relationship and it is represented by double diamond.



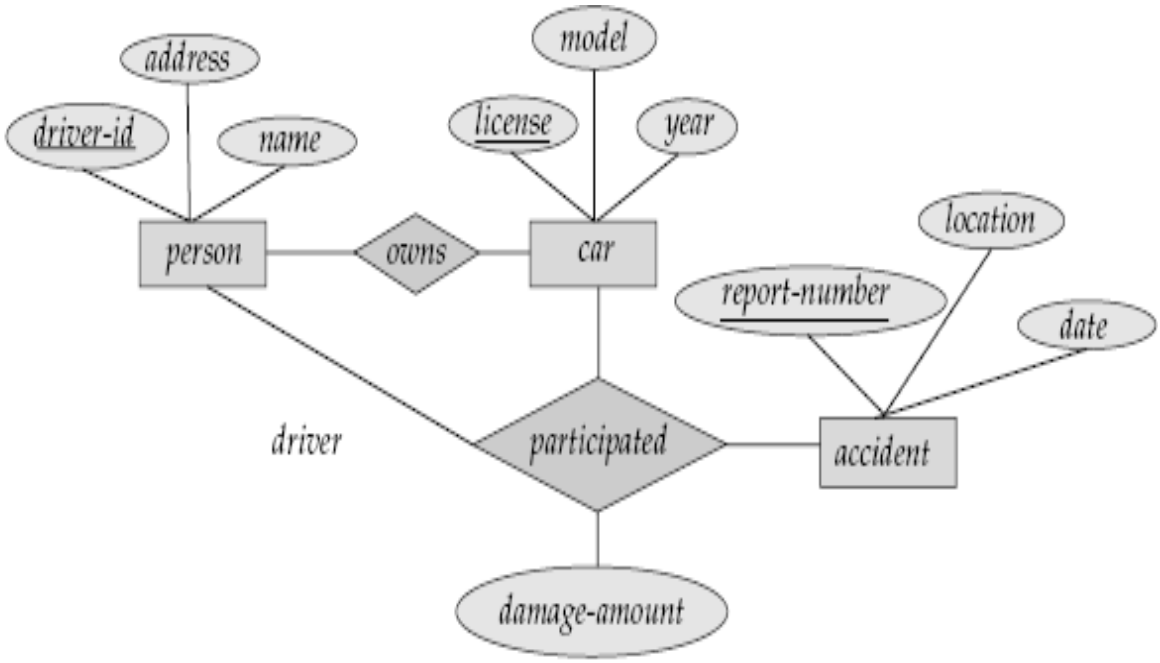
Symbols use ER- diagram:



E-R Diagram for Hospital:



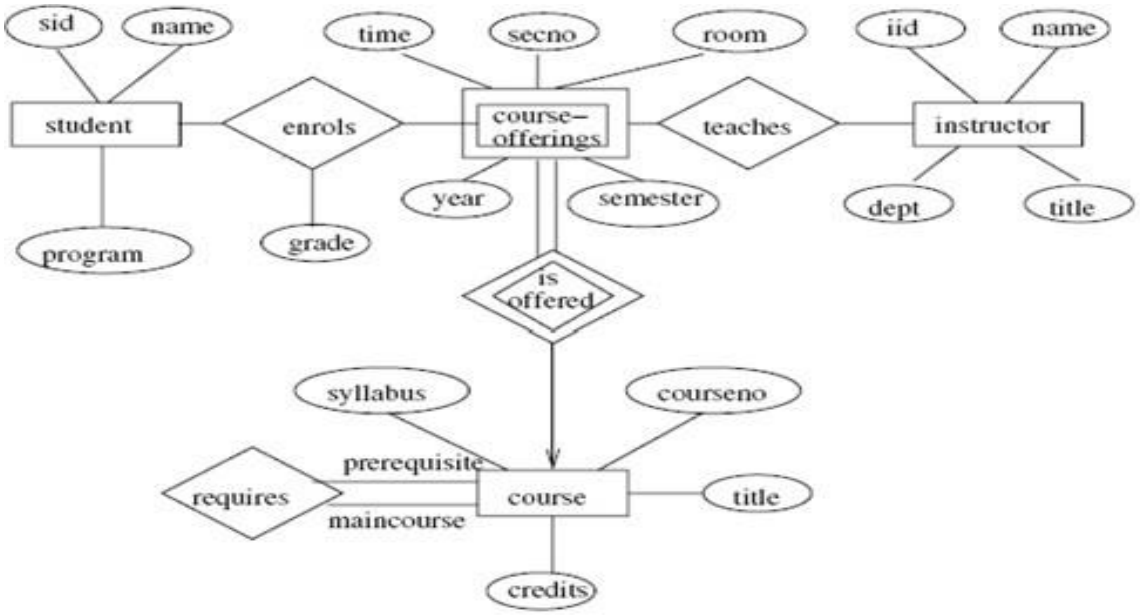
E-R Diagram for Car Insurance Company:



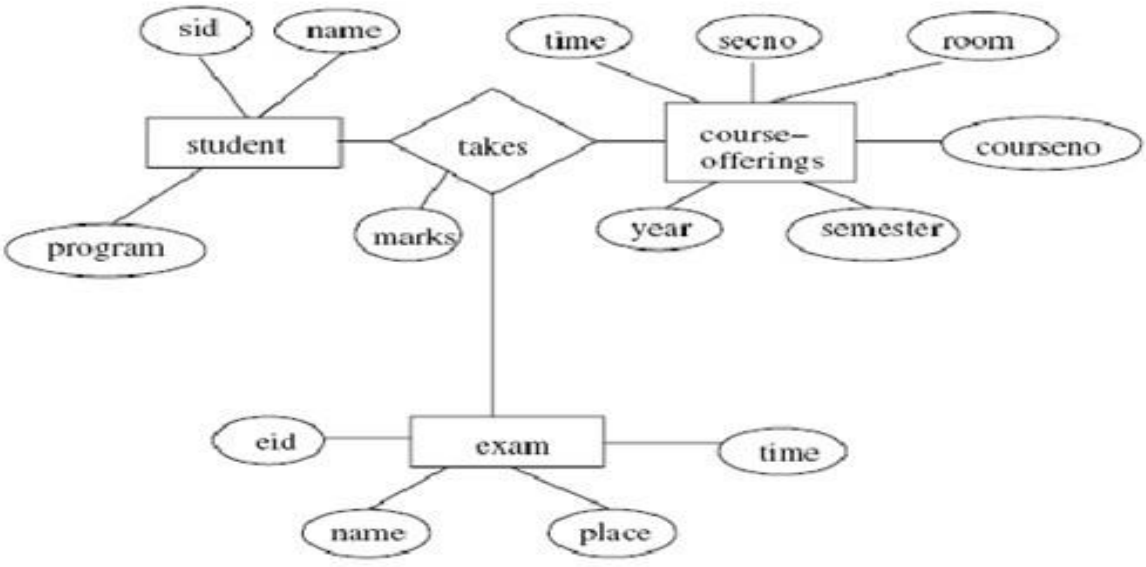
E-R Diagram for University and marks Database:

A university registrar's office maintains data about the following entities: (a) courses, including number, title, credits, syllabus, and prerequisites; (b) course offerings, including course number, year, semester, section number, instructor(s), timings, and classroom; (c) students, including student-id, name, and program; and (d) instructors, including identification number, name, department, and title. Further, the enrollment of students in courses and grades awarded to students in each course they are enrolled for must be appropriately modeled.

Construct an E-R diagram for the registrar's office. Document all assumptions that you make about the mapping constraints.

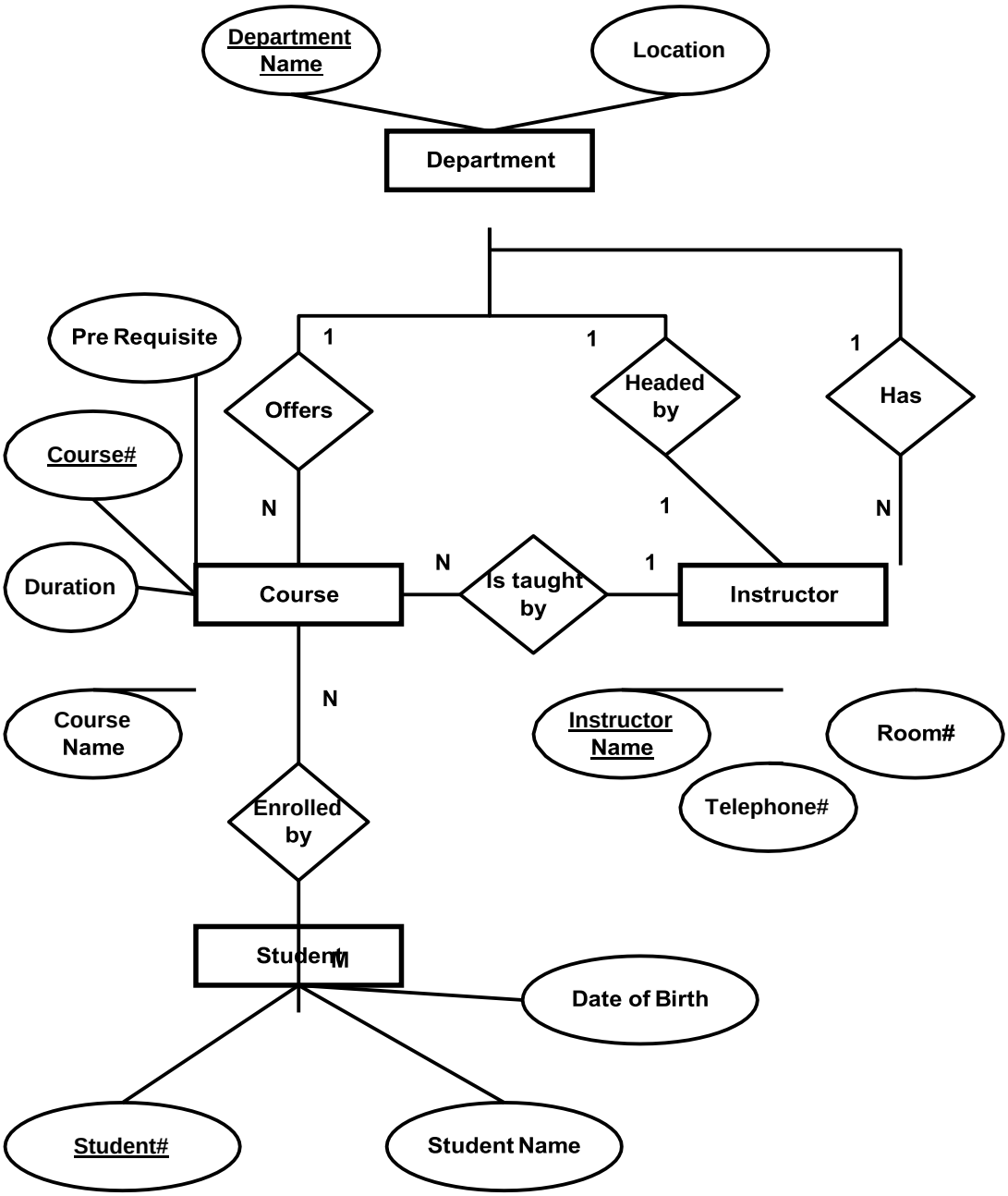


E-R diagram for a university.



E-R diagram for marks database.

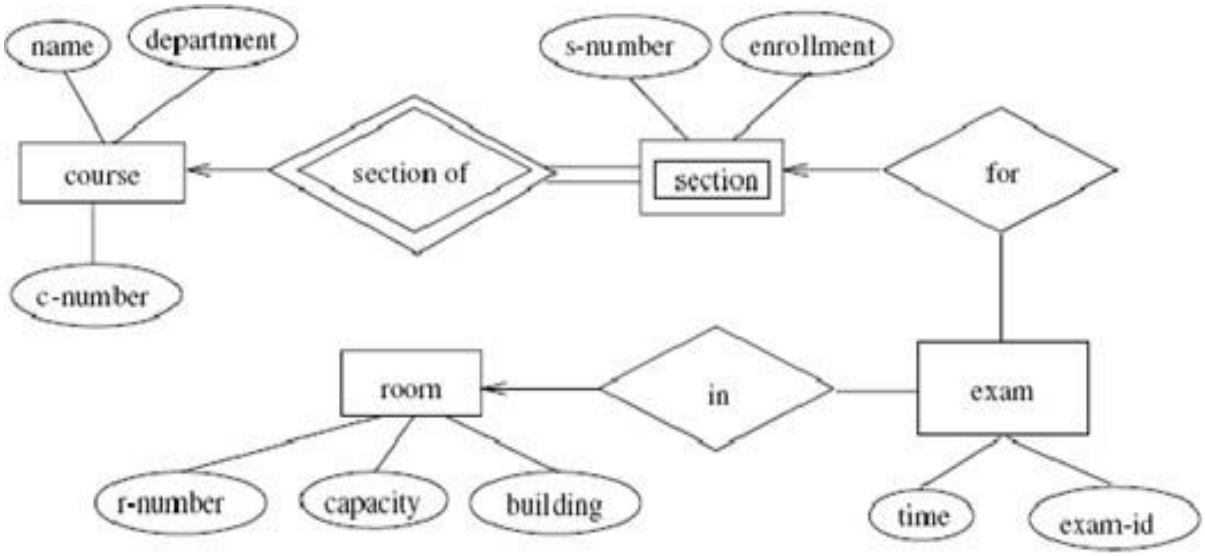
ER Model for the University Database



E-R diagram for student examination:

Consider a university database for the scheduling of classrooms for final exams. This database could be modeled as the single entity set *exam*, with attributes *course-name*, *section-number*, *room-number*, and *time*. Alternatively, one or more additional entity sets could be defined, along with relationship sets to replace some of the attributes of the *exam* entity set, as

- *course* with attributes *name*, *department*, and *c-number*
- *section* with attributes *s-number* and *enrollment*, and dependent as a weak entity set on *course*
- *room* with attributes *r-number*, *capacity*, and *building*

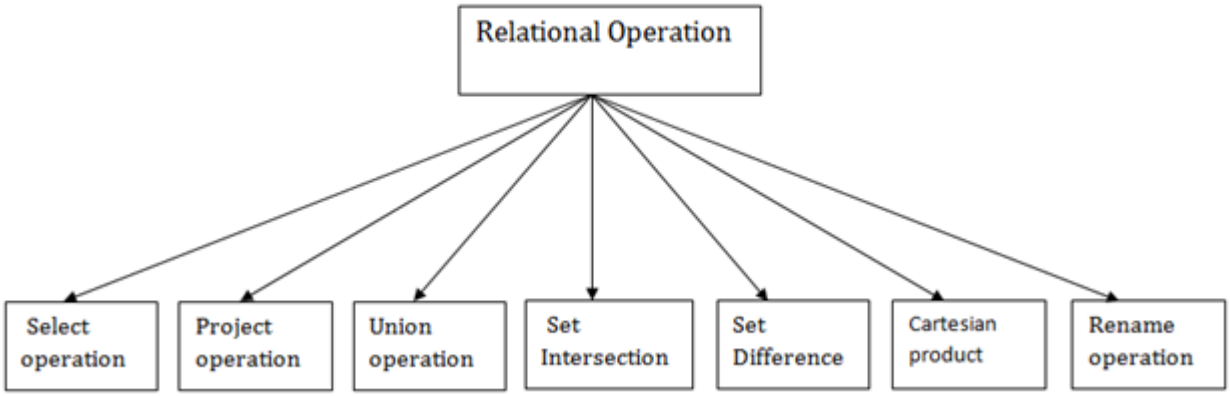


E-R diagram for exam scheduling.

Relational Algebra:

Relational algebra is a procedural query language. It gives a step by step process to obtain the result of the query. It uses operators to perform queries.

Types:



1.Select Operation:

- The select operation selects tuples that satisfy a given predicate.
- It is denoted by sigma (σ).

1. Notation: σ p(r)

Where:

σ is used for selection prediction
r is used for relation
p is used as a propositional logic formula which may use connectors like: AND OR and NOT. These relational can use as relational operators like =, ≠, ≥, <, >, ≤.

For example: LOAN Relation

BRANCH_NAME	LOAN_NO	AMOUNT
Downtown	L-17	1000
Redwood	L-23	2000
Perryride	L-15	1500
Downtown	L-14	1500
Mianus	L-13	500
Roundhill	L-11	900
Perryride	L-16	1300

Input:

1. σ BRANCH_NAME="perryride" (LOAN)

Output:

BRANCH_NAME	LOAN_NO	AMOUNT
Perryride	L-15	1500
Perryride	L-16	1300

2. Project Operation:

- This operation shows the list of those attributes that we wish to appear in the result. Rest of the attributes are eliminated from the table.
- It is denoted by ∏.

1. Notation: ∏ A1, A2, An (r)

Where

A1, A2, A3 is used as an attribute name of relation r.

Example: CUSTOMER RELATION

NAME	STREET	CITY

Jones	Main	Harrison
Smith	North	Rye
Hays	Main	Harrison
Curry	North	Rye
Johnson	Alma	Brooklyn
Brooks	Senator	Brooklyn

Input:

1. ∏ NAME, CITY (CUSTOMER)

Output:

NAME	CITY
Jones	Harrison
Smith	Rye
Hays	Harrison
Curry	Rye
Johnson	Brooklyn
Brooks	Brooklyn

3. Union Operation:

- Suppose there are two tuples R and S. The union operation contains all the tuples that are either in R or S or both in R & S.
- It eliminates the duplicate tuples. It is denoted by ∪.

1. Notation: R ∪ S

A union operation must hold the following condition:

- R and S must have the attribute of the same number.
- Duplicate tuples are eliminated automatically.

Example:

DEPOSITOR RELATION

CUSTOMER_NAME	ACCOUNT_NO
Johnson	A-101
Smith	A-121
Mayes	A-321
Turner	A-176
Johnson	A-273
Jones	A-472
Lindsay	A-284

BORROW RELATION

CUSTOMER_NAME	LOAN_NO
Jones	L-17
Smith	L-23
Hayes	L-15
Jackson	L-14
Curry	L-93
Smith	L-11
Williams	L-17

Input:

1. ∏ CUSTOMER_NAME (BORROW) ∪ ∏ CUSTOMER_NAME (DEPOSITOR)

Output:

CUSTOMER_NAME
Johnson
Smith
Hayes
Turner
Jones
Lindsay
Jackson
Curry
Williams
Mayes

4. Set Intersection:

- Suppose there are two tuples R and S. The set intersection operation contains all tuples that are in both R & S.
- It is denoted by intersection ∩.

1. Notation: R ∩ S

Example: Using the above DEPOSITOR table and BORROW table

Input:

1. ∏ CUSTOMER_NAME (BORROW) ∩ ∏ CUSTOMER_NAME (DEPOSITOR)

Output:

CUSTOMER_NAME
Smith
Jones

5. Set Difference:

- Suppose there are two tuples R and S. The set intersection operation contains all tuples that are in R but not in S.
- It is denoted by intersection minus (-).

1. Notation: R - S

Example: Using the above DEPOSITOR table and BORROW table

Input:

1. ∏ CUSTOMER_NAME (BORROW) - ∏ CUSTOMER_NAME (DEPOSITOR)

Output:

CUSTOMER_NAME
Jackson
Hayes
Willians
Curry

6. Cartesian product

- The Cartesian product is used to combine each row in one table with each row in the other table. It is also known as a cross product.
- It is denoted by X.

1. Notation: E X D

Example:

EMPLOYEE

EMP_ID	EMP_NAME	EMP_DEPT
1	Smith	A
2	Harry	C
3	John	B

DEPARTMENT

DEPT_NO	DEPT_NAME
A	Marketing
B	Sales
C	Legal

Input:

1. EMPLOYEE X DEPARTMENT

Output:

EMP_ID	EMP_NAME	EMP_DEPT	DEPT_NO	DEPT_NAME
1	Smith	A	A	Marketing
1	Smith	A	B	Sales
1	Smith	A	C	Legal
2	Harry	C	A	Marketing
2	Harry	C	B	Sales
2	Harry	C	C	Legal
3	John	B	A	Marketing
3	John	B	B	Sales
3	John	B	C	Legal

7. Rename Operation:

The rename operation is used to rename the output relation. It is denoted by rho (ρ).

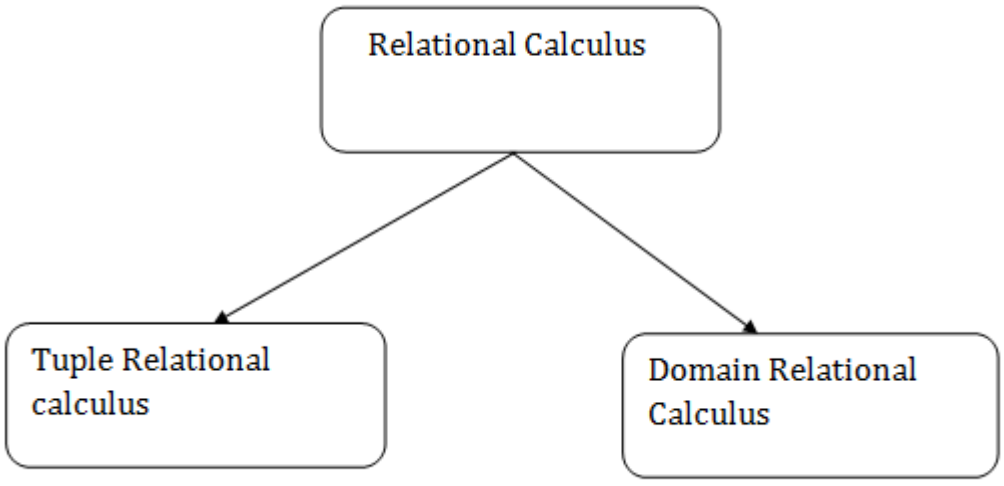
Example: We can use the rename operator to rename STUDENT relation to STUDENT1.

1. ρ(STUDENT1, STUDENT)

Relational Calculus

- Relational calculus is a non-procedural query language. In the non-procedural query language, the user is concerned with the details of how to obtain the end results.
- The relational calculus tells what to do but never explains how to do.

Types of Relational calculus:



1. Tuple Relational Calculus (TRC)

- The tuple relational calculus is specified to select the tuples in a relation. In TRC, filtering variable uses the tuples of a relation.
- The result of the relation can have one or more tuples.

Notation:

1. {T | P (T)} or {T | Condition (T)}

Where

T is the resulting tuples

P(T) is the condition used to fetch T.

For example:

1. { T.name | Author(T) AND T.article = 'database' }

OUTPUT: This query selects the tuples from the AUTHOR relation. It returns a tuple with 'name' from Author who has written an article on 'database'.

TRC (tuple relation calculus) can be quantified. In TRC, we can use Existential (∃) and Universal Quantifiers (∀).

For example:

1. { R| ∃T ∈ Authors(T.article='database' AND R.name=T.name)}

Output: This query will yield the same result as the previous one.

2. Domain Relational Calculus (DRC)

- The second form of relation is known as Domain relational calculus. In domain relational calculus, filtering variable uses the domain of attributes.
- Domain relational calculus uses the same operators as tuple calculus. It uses logical connectives ∧ (and), ∨ (or) and ¬ (not).
- It uses Existential (∃) and Universal Quantifiers (∀) to bind the variable.

Notation:

1. { a1, a2, a3, ..., an | P (a1, a2, a3, ... ,an)}

Where

a1, a2 are attributes
P stands for formula built by inner attributes

For example:

1. {< article, page, subject > | ∈ javatpoint ∧ subject = 'database'}

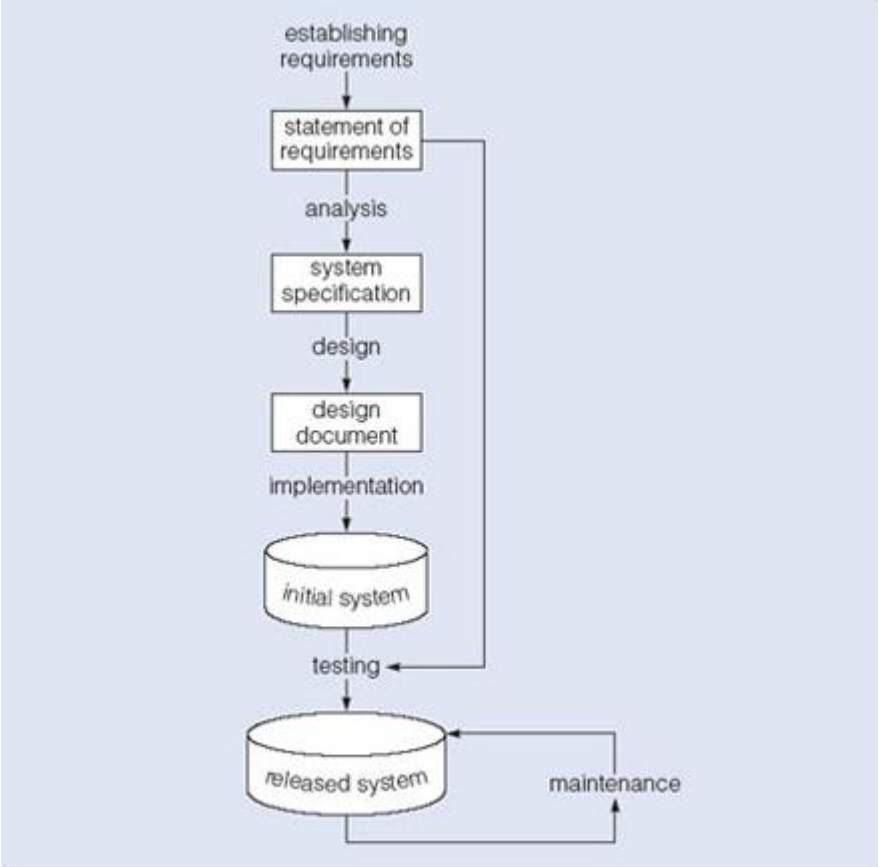
Output: This query will yield the article, page, and subject from the relational javatpoint, where the subject is a database.

QBE (Query By Example): Stands for "Query By Example." QBE is a feature included with various database applications that provides a user-friendly method of running database queries. Typically without QBE, a user must write input commands using correct SQL (Structured Query Language) syntax. This is a standard language that nearly all database programs support. However, if the syntax is slightly incorrect the query may return the wrong results or may not run at all.

The Query By Example feature provides a simple interface for a user to enter queries. Instead of writing an entire SQL command, the user can just fill in blanks or select items to define the query she wants to perform. For example, a user may want to select an entry from a table called "Table1" with an ID of 123. Using SQL, the user would need to input the command, "SELECT * FROM Table1 WHERE ID = 123". The QBE interface may allow the user to just click on Table1, type in "123" in the ID field and click "Search."

QBE is offered with most database programs, though the interface is often different between applications. For example, Microsoft Access has a QBE interface known as "Query Design View" that is completely graphical. The phpMyAdmin application used with MySQL, offers a Web-based interface where users can select a query operator and fill in blanks with search terms. Whatever QBE implementation is provided with a program, the purpose is the same – to make it easier to run database queries and to avoid the frustrations of SQL errors.

Database Development Life Cycle(DDLC):



Database development is just one part of the much wider field of software engineering, the process of developing and maintaining software. A core aspect of software engineering is the subdivision of the development process into a series of phases, or steps, each of which focuses on one aspect of the development. The collection of these steps is sometimes referred to as a development life cycle.

- Establishing requirements involves consultation with, and agreement among, stakeholders as to what they want of a system, expressed as a statement of requirements.
- Analysis starts by considering the statement of requirements and finishes by producing a system specification. The specification is a formal representation of what a system should do, expressed in terms that are independent of how it may be realised.
- Design begins with a system specification and produces design documents, and provides a detailed description of how a system should be constructed.
- Implementation is the construction of a computer system according to a given design document and taking account of the environment in which the system will be operating (for example specific hardware or software available for the development). Implementation may be staged, usually with an initial system than can be validated and tested before a final system is released for use.
- Testing compares the implemented system against the design documents and requirements specification and produces an acceptance report or, more usually, a list of errors and bugs that require a review of the analysis, design and implementation processes to correct (testing is usually the task that leads to the waterfall model iterating through the life cycle).
- Maintenance involves dealing with changes in the requirements, or the implementation environment, bug fixing or porting of the system to new environments (for example migrating a system from a standalone PC to a UNIX workstation or a networked environment). Since maintenance involves the analysis of the changes required, design of a solution, implementation and testing of that solution over the lifetime of a maintained software system, the waterfall life cycle will be repeatedly revisited.

Automated Design Tools: There are different automated design tools they are

1.Diagramming: this allows the designer to draw a conceptual schema diagram in some tool specific notations.

2.Model Mapping: this implements mapping algorithms, this mapping is a system specific most tools generated schemas in SQL DDL for Oracle.

3.Design Normalization: This utilizes a set of functional dependencies that are supplied at the conceptual design.

4.Trade off analysis: A tool should be present the designer with adequate analysis

5.Display of Design results: Design results such as schemas are often displayed in diagrammatic form. Diagrams are not easy to generate automatically so we can slow the design in form the tables.

6.Design Verification: Its purpose is to verify that the resulting design satisfies the initial requirements.

#1) Visual Paradigm ERD Tools. designing of the database by following a powerful approach of Entity Relationship Diagram Tools (ERD)

#2) Vertabelo. for designing your database online. This uses the power of Visual Modelling which reduces the pain of manual creation of a dozen of tables in the database.

#3) Lucidchart. Database Design tools which will help you in quickly creating database diagrams online with its collaborative database design tools.

#4) SQL Server Database Modeler. This is one of the commonly used Database Design tools that is used to design your database online by importing the existing database. This follows the concepts of Forward Engineering and Reverse Engineering.

#5) DeZign for Databases. Database Designing tools which supports the feature of Data Modeling for database professionals. This is robust and it is very easy to use this tool.

- #6) Erwin Data Modeler. database designing tools which support the feature of Data Modeling by unlocking the value out of the Enterprise Data....
- #7) Aqua Data Studio ER Modeler. This supports Entity-Relationship Modeling by providing a feature to design physical models for all major Relational Database Management Systems. ...
- #8) DbWrench. This tool supports synchronization along with designing of the databases with the guarantee to save the time of the users in handling multiple database tasks. This tool also has the feature to forward and reverse engineer the databases.

Join Strategies:

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

Example:

EMPLOYEE

EMP_CODE	EMP_NAME
101	Stephan
102	Jack
103	Harry

SALARY

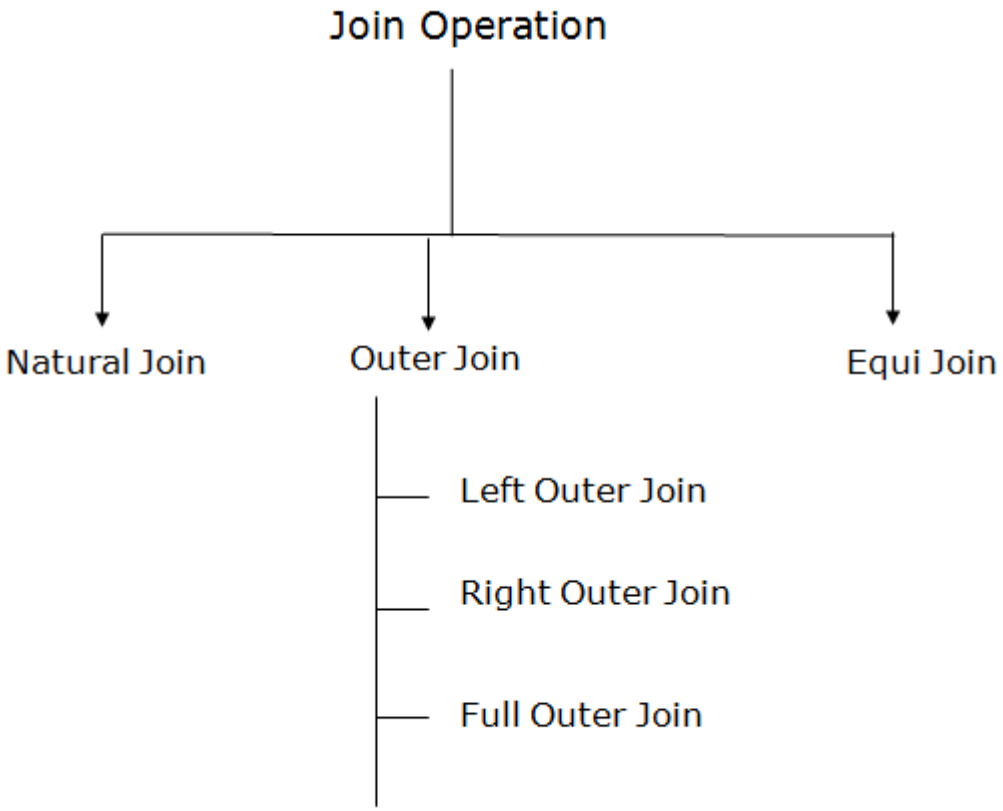
EMP_CODE	SALARY
101	50000
102	30000
103	25000

1. Operation: (EMPLOYEE ⋈ SALARY)

Result:

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

Types of Join operations:



1. Natural Join:

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

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

Input:

1. ∏EMP_NAME, SALARY (EMPLOYEE ⋈ SALARY)

Output:

EMP_NAME	SALARY
Stephan	50000
Jack	30000
Harry	25000

2. Outer Join:

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

Example:

EMPLOYEE

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

FACT_WORKERS

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

Input:

1. (EMPLOYEE ⋈ FACT_WORKERS)

Output:

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

An outer join is basically of three types:

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

a. Left outer join:

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

- It is denoted by ⋈.

Example: Using the above EMPLOYEE table and FACT_WORKERS table

Input:

1. EMPLOYEE ⋈ FACT_WORKERS

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

b. Right outer join:

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

Example: Using the above EMPLOYEE table and FACT_WORKERS Relation

Input:

1. EMPLOYEE ⋉ FACT_WORKERS

Output:

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

c. Full outer join:

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

Example: Using the above EMPLOYEE table and FACT_WORKERS table

Input:

1. EMPLOYEE ⋉ FACT_WORKERS

Output:

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

3. Equi join:

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

Example:

CUSTOMER RELATION

CLASS_ID	NAME
1	John
2	Harry
3	Jackson

PRODUCT

PRODUCT_ID	CITY
1	Delhi
2	Mumbai
3	Noida

Input:

1. CUSTOMER ⋈ PRODUCT

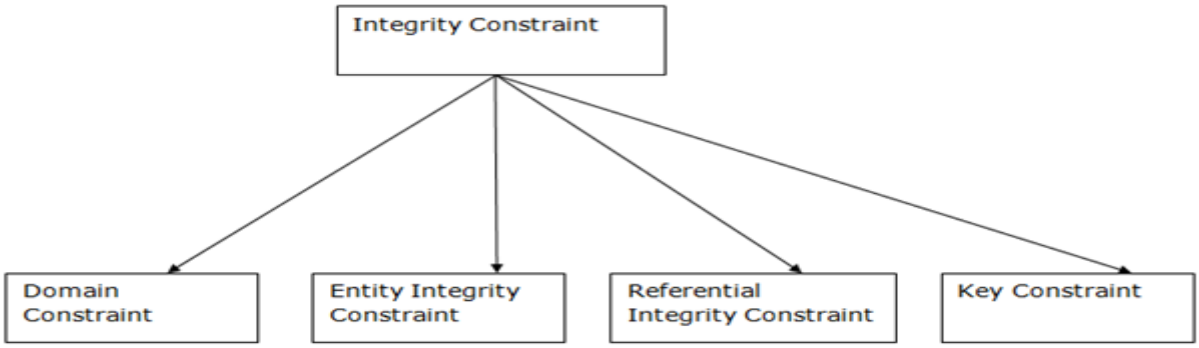
Output:

CLASS_ID	NAME	PRODUCT_ID	CITY
1	John	1	Delhi
2	Harry	2	Mumbai
3	Harry	3	Noida

Integrity Constraints

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

Types of Integrity Constraint



1. Domain constraints

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

Example:

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

Not allowed. Because AGE is an integer attribute

2. Entity integrity constraints

- The entity integrity constraint states that primary key value can't be null.

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

Example:

EMPLOYEE

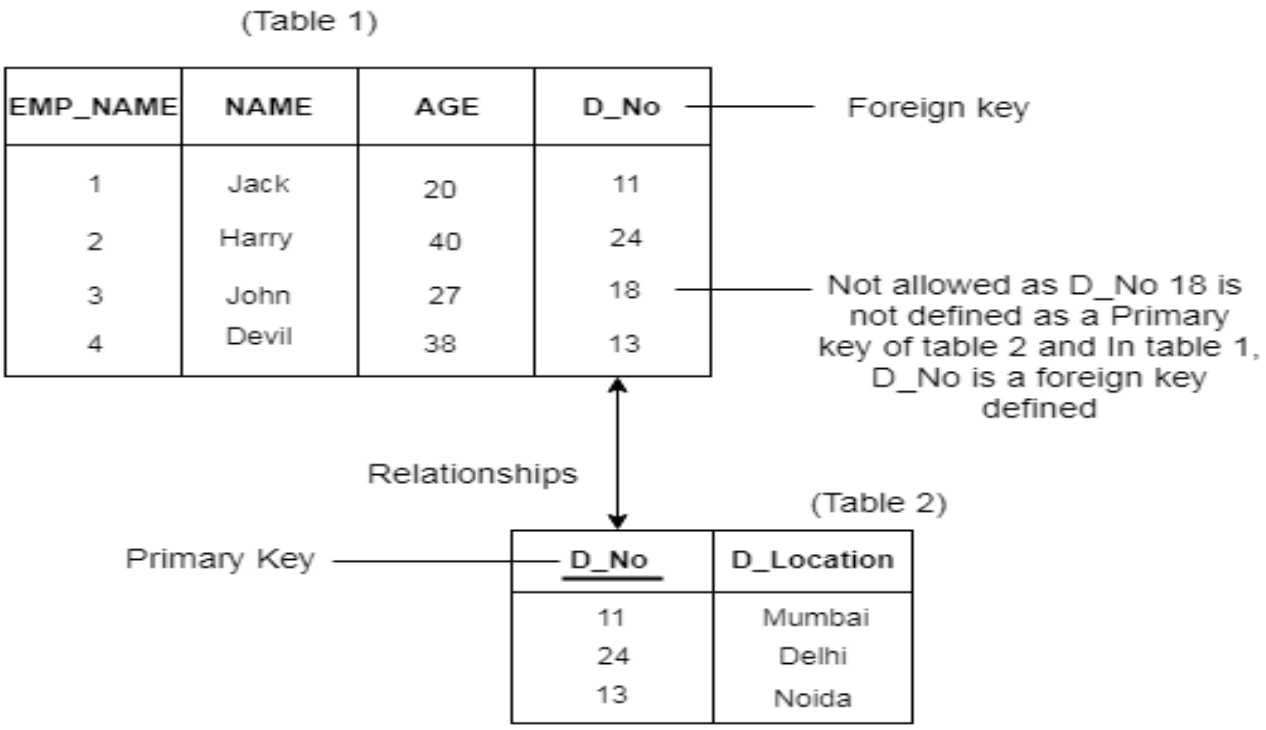
EMP_ID	EMP_NAME	SALARY
123	Jack	30000
142	Harry	60000
164	John	20000
	Jackson	27000

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

3. Referential Integrity Constraints

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

Example:



4. Key constraints

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

Example:

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

Not allowed. Because all row must be unique

Functional Dependency

The functional dependency is a relationship that exists between two attributes. It typically exists between the primary key and non-key attribute within a table.

1. X → Y

The left side of FD is known as a determinant; the right side of the production is known as a dependent.

For example:

Assume we have an employee table with attributes: Emp_Id, Emp_Name, Emp_Address.

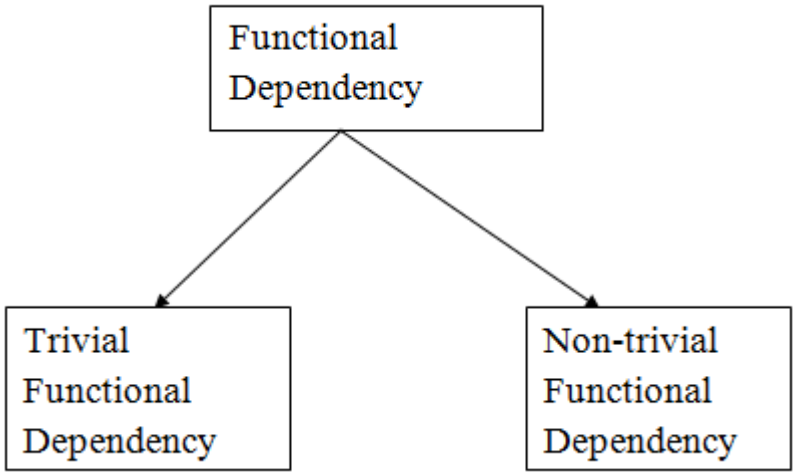
Here Emp_Id attribute can uniquely identify the Emp_Name attribute of employee table because if we know the Emp_Id, we can tell that employee name associated with it.

Functional dependency can be written as:

1. Emp_Id → Emp_Name

We can say that Emp_Name is functionally dependent on Emp_Id.

Types of Functional dependency



1. Trivial functional dependency

- A → B has trivial functional dependency if B is a subset of A.
- The following dependencies are also trivial like: A → A, B → B

Example:

1. Consider a table with two columns Employee_Id and Employee_Name.
2. {Employee_id, Employee_Name} → Employee_Id is a trivial functional dependency as
3. Employee_Id is a subset of {Employee_Id, Employee_Name}.
4. Also, Employee_Id → Employee_Id and Employee_Name → Employee_Name are trivial dependencies too.

2. Non-trivial functional dependency

- A → B has a non-trivial functional dependency if B is not a subset of A.
- When A intersection B is NULL, then A → B is called as complete non-trivial.

Example:

1. ID → Name,
2. Name → DOB

Inference Rule (IR):

- The Armstrong's axioms are the basic inference rule.
- Armstrong's axioms are used to conclude functional dependencies on a relational database.
- The inference rule is a type of assertion. It can apply to a set of FD(functional dependency) to derive other FD.
- Using the inference rule, we can derive additional functional dependency from the initial set.

The Functional dependency has 6 types of inference rule:

1. Reflexive Rule (IR_1)

In the reflexive rule, if Y is a subset of X, then X determines Y.

1. If $X \supseteq Y$ then $X \rightarrow Y$

Example:

1. X = {a, b, c, d, e}
2. Y = {a, b, c}

2. Augmentation Rule (IR_2)

The augmentation is also called as a partial dependency. In augmentation, if X determines Y, then XZ determines YZ for any Z.

1. If $X \rightarrow Y$ then $XZ \rightarrow YZ$

Example:

1. For R(ABCD), if $A \rightarrow B$ then $AC \rightarrow BC$

3. Transitive Rule (IR_3)

In the transitive rule, if X determines Y and Y determine Z, then X must also determine Z.

1. If $X \rightarrow Y$ and $Y \rightarrow Z$ then $X \rightarrow Z$

4. Union Rule (IR_4)

Union rule says, if X determines Y and X determines Z, then X must also determine Y and Z.

1. If $X \rightarrow Y$ and $X \rightarrow Z$ then $X \rightarrow YZ$

Proof:

1. $X \rightarrow Y$ (given)
2. $X \rightarrow Z$ (given)
3. $X \rightarrow XY$ (using IR_2 on 1 by augmentation with X. Where XX = X)
4. $XY \rightarrow YZ$ (using IR_2 on 2 by augmentation with Y)
5. $X \rightarrow YZ$ (using IR_3 on 3 and 4)

5. Decomposition Rule (IR_5)

Decomposition rule is also known as project rule. It is the reverse of union rule.

This Rule says, if X determines Y and Z, then X determines Y and X determines Z separately.

1. If $X \rightarrow YZ$ then $X \rightarrow Y$ and $X \rightarrow Z$

Proof:

1. X → YZ (given)
2. YZ → Y (using IR_1 Rule)
3. X → Y (using IR_3 on 1 and 2)

6. Pseudo transitive Rule (IR_6)

In Pseudo transitive Rule, if X determines Y and YZ determines W, then XZ determines W.

1. If X → Y and YZ → W then XZ → W

Proof:

1. X → Y (given)
2. WY → Z (given)
3. WX → WY (using IR_2 on 1 by augmenting with W)
4. WX → Z (using IR_3 on 3 and 2)

Closure Of Functional Dependency :

Introduction

- The Closure Of Functional Dependency means the complete set of all possible attributes that can be functionally derived from given functional dependency using the inference rules known as Armstrong's Rules.
- If "F" is a functional dependency then closure of functional dependency can be denoted using "$\{F\}^+$".
- There are three steps to calculate closure of functional dependency. These are:

Step-1 : Add the attributes which are present on Left Hand Side in the original functional dependency.
Step-2 : Now, add the attributes present on the Right Hand Side of the functional dependency.
Step-3 : With the help of attributes present on Right Hand Side, check the other attributes that can be derived from the other given functional dependencies. Repeat this process until all the possible attributes which can be derived are added in the closure.

Closure Of Functional Dependency :

Examples

Example-1 : Consider the table student_details having (Roll_No, Name,Marks, Location) as the attributes and having two functional dependencies.
FD1 : Roll_No Name, Marks
FD2 : Name Marks, Location

Now, We will calculate the closure of all the attributes present in the relation using the three steps mentioned below.

Step-1 : Add attributes present on the LHS of the first functional dependency to the closure.
$\{Roll\_no\}^+ = \{Roll\_No\}$

Step-2 : Add attributes present on the RHS of the original functional dependency to the closure.
$\{Roll\_no\}^+ = \{Roll\_No, Marks\}$

Step-3 : Add the other possible attributes which can be derived using attributes present on the RHS of the closure.

So Roll_No attribute cannot functionally determine any attribute but Name attribute can determine other attributes

Marks and Location using 2nd Functional Dependency(Name Marks, Location).
Therefore, complete closure of Roll_No will be :

$\{Roll\_no\}^+ = \{Roll\_No, Marks, Name, Location\}$

Similarly, we can calculate closure for other attributes too i.e “Name”.

Step-1 : Add attributes present on the LHS of the functional dependency to the closure.
$\{Name\}^+ = \{Name\}$

Step-2 : Add the attributes present on the RHS of the functional dependency to the closure.
$\{Name\}^+ = \{Name, Marks, Location\}$

Step-3 : Since, we don’t have any functional dependency where “Marks or Location” attribute is functionally deter

other attribute , we cannot add more attributes to the closure. Hence complete closure of Name would be :
$\{Name\}^+ = \{Name, Marks, Location\}$

NOTE : We don’t have any Functional dependency where marks and location can functionally determine any attrib

for those attributes we can only add the attributes themselves in their closures. Therefore,
$\{Marks\}^+ = \{Marks\}$
and
$\{Location\}^+ = \{ Location\}$

Example-2 : Consider a relation R(A,B,C,D,E) having below mentioned functional dependencies.

FD1 : A BC
FD2 : C B
FD3 : D E
FD4 : E D

Now, we need to calculate the closure of attributes of the relation R. The closures will be:

$\{A\}^+ = \{A, B, C\}$

$\{B\}^+ = \{B\}$

$\{C\}^+ = \{B, C\}$

$\{D\}^+ = \{D, E\}$

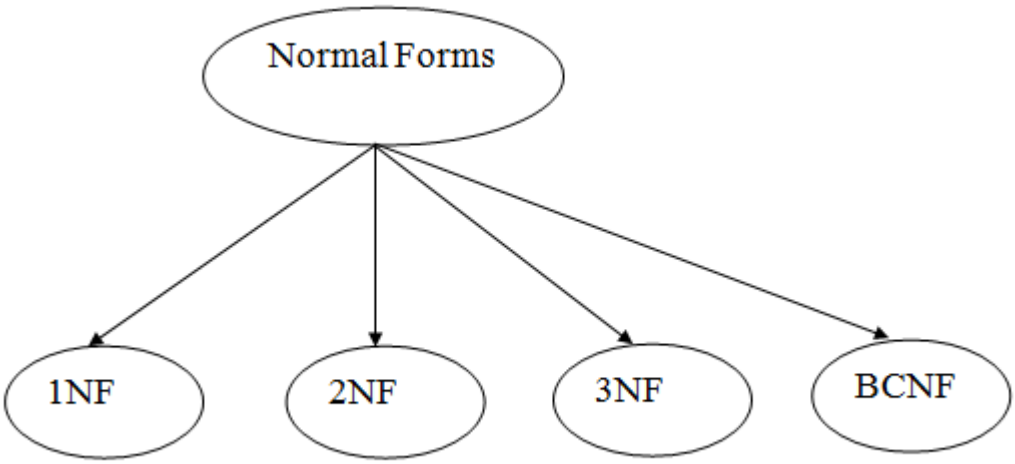
$\{E\}^+ = \{E\}$

Normalization

- Normalization is the process of organizing the data in the database.
- Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate the undesirable characteristics like Insertion, Update and Deletion Anomalies.
- Normalization divides the larger table into the smaller table and links them using relationship.
- The normal form is used to reduce redundancy from the database table.

Types of Normal Forms

There are the four types of normal forms:



Normal Form	Description
1NF	A relation is in 1NF if it contains an atomic value.
2NF	A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key.
3NF	A relation will be in 3NF if it is in 2NF and no transition dependency exists.
4NF	A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency.
5NF	A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.

First Normal Form (1NF)

- A relation will be 1NF if it contains an atomic value.
- It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.
- First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

Example: Relation EMPLOYEE is not in 1NF because of multi-valued attribute EMP_PHONE.

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385, 9064738238	UP
20	Harry	8574783832	Bihar

12	Sam	7390372389, 8589830302	Punjab

The decomposition of the EMPLOYEE table into 1NF has been shown below:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385	UP
14	John	9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389	Punjab
12	Sam	8589830302	Punjab

Second Normal Form (2NF)

- In the 2NF, relational must be in 1NF.
- In the second normal form, all non-key attributes are fully functional dependent on the primary key

Example: Let's assume, a school can store the data of teachers and the subjects they teach. In a school, a teacher can teach more than one subject.

TEACHER table

TEACHER_ID	SUBJECT	TEACHER_AGE
25	Chemistry	30
25	Biology	30
47	English	35
83	Math	38
83	Computer	38

In the given table, non-prime attribute TEACHER_AGE is dependent on TEACHER_ID which is a proper subset of a candidate key. That's why it violates the rule for 2NF.

To convert the given table into 2NF, we decompose it into two tables:

TEACHER_DETAIL table:

TEACHER_ID	TEACHER_AGE
25	30

47	35
83	38

TEACHER_SUBJECT table:

TEACHER_ID	SUBJECT
25	Chemistry
25	Biology
47	English
83	Math
83	Computer

Third Normal Form (3NF)

- A relation will be in 3NF if it is in 2NF and not contain any transitive partial dependency.
- 3NF is used to reduce the data duplication. It is also used to achieve the data integrity.
- If there is no transitive dependency for non-prime attributes, then the relation must be in third normal form.

A relation is in third normal form if it holds at least one of the following conditions for every non-trivial function dependency X → Y.

1. X is a super key.
2. Y is a prime attribute, i.e., each element of Y is part of some candidate key.

   Example:

   EMPLOYEE_DETAIL table:

EMP_ID	EMP_NAME	EMP_ZIP	EMP_STATE	EMP_CITY
222	Harry	201010	UP	Noida
333	Stephan	02228	US	Boston
444	Lan	60007	US	Chicago
555	Katharine	06389	UK	Norwich
666	John	462007	MP	Bhopal

   Super key in the table above:

   1. {EMP_ID}, {EMP_ID, EMP_NAME}, {EMP_ID, EMP_NAME, EMP_ZIP}....so on

Candidate key: {EMP_ID}

Non-prime attributes: In the given table, all attributes except EMP_ID are non-prime.

Here, EMP_STATE & EMP_CITY dependent on EMP_ZIP and EMP_ZIP dependent on EMP_ID. The non-prime attributes (EMP_STATE, EMP_CITY) transitively dependent on super key(EMP_ID). It violates the rule of third normal form.

That's why we need to move the EMP_CITY and EMP_STATE to the new <EMPLOYEE_ZIP> table, with EMP_ZIP as a Primary key.

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_ZIP
222	Harry	201010
333	Stephan	02228
444	Lan	60007
555	Katharine	06389
666	John	462007

EMPLOYEE_ZIP table:

EMP_ZIP	EMP_STATE	EMP_CITY
201010	UP	Noida
02228	US	Boston
60007	US	Chicago
06389	UK	Norwich
462007	MP	Bhopal

Boyce Codd normal form (BCNF)

- BCNF is the advance version of 3NF. It is stricter than 3NF.
- A table is in BCNF if every functional dependency X → Y, X is the super key of the table.
- For BCNF, the table should be in 3NF, and for every FD, LHS is super key.

Example: Let's assume there is a company where employees work in more than one department.

EMPLOYEE table:

EMP_ID EMP_COUNTRY EMP_DEPT DEPT_TYPE EMP_DEPT_NO

264	India	Designing	D394	283
264	India	Testing	D394	300
364	UK	Stores	D283	232
364	UK	Developing	D283	549

In the above table Functional dependencies are as follows:

1. EMP_ID → EMP_COUNTRY
2. EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Candidate key: {EMP-ID, EMP-DEPT}

The table is not in BCNF because neither EMP_DEPT nor EMP_ID alone are keys.

To convert the given table into BCNF, we decompose it into three tables:

EMP_COUNTRY table:

EMP_ID	EMP_COUNTRY
264	India
264	India

EMP_DEPT table:

EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
Designing	D394	283
Testing	D394	300
Stores	D283	232
Developing	D283	549

EMP_DEPT_MAPPING table:

EMP_ID	EMP_DEPT
D394	283
D394	300
D283	232
D283	549

Functional dependencies:

1. EMP_ID → EMP_COUNTRY
2. EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Candidate keys:

For the first table: EMP_ID
For the second table: EMP_DEPT
For the third table: {EMP_ID, EMP_DEPT}

Now, this is in BCNF because left side part of both the functional dependencies is a key.

Multivalued Dependency

- Multivalued dependency occurs when two attributes in a table are independent of each other but, both depend on a third attribute.
- A multivalued dependency consists of at least two attributes that are dependent on a third attribute that's why it always requires at least three attributes.

Example: Suppose there is a bike manufacturer company which produces two colors(white and black) of each model every year.

BIKE_MODEL	MANUF_YEAR	COLOR
M2011	2008	White
M2001	2008	Black
M3001	2013	White
M3001	2013	Black
M4006	2017	White
M4006	2017	Black

Here columns COLOR and MANUF_YEAR are dependent on BIKE_MODEL and independent of each other.

In this case, these two columns can be called as multivalued dependent on BIKE_MODEL. The representation of these dependencies is shown below:

1. BIKE_MODEL → → MANUF_YEAR
2. BIKE_MODEL → → COLOR

This can be read as "BIKE_MODEL multidetermined MANUF_YEAR" and "BIKE_MODEL multidetermined COLOR".

Fourth normal form (4NF)

- A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency.

- For a dependency A → B, if for a single value of A, multiple values of B exists, then the relation will be a multi-valued dependency.

Example

STUDENT

STU_ID	COURSE	HOBBY
21	Computer	Dancing
21	Math	Singing
34	Chemistry	Dancing
74	Biology	Cricket
59	Physics	Hockey

The given STUDENT table is in 3NF, but the COURSE and HOBBY are two independent entity. Hence, there is no relationship between COURSE and HOBBY.

In the STUDENT relation, a student with STU_ID, 21 contains two courses, Computer and Math and two hobbies, Dancing and Singing. So there is a Multi-valued dependency on STU_ID, which leads to unnecessary repetition of data.

So to make the above table into 4NF, we can decompose it into two tables:

STUDENT_COURSE

STU_ID	COURSE
21	Computer
21	Math
34	Chemistry
74	Biology
59	Physics

STUDENT_HOBBY

STU_ID	HOBBY
21	Dancing
21	Singing
34	Dancing

74 Cricket

59 Hockey

Join Dependency

- Join decomposition is a further generalization of Multivalued dependencies.
- If the join of R1 and R2 over C is equal to relation R, then we can say that a join dependency (JD) exists.
- Where R1 and R2 are the decompositions R1(A, B, C) and R2(C, D) of a given relations R (A, B, C, D).
- Alternatively, R1 and R2 are a lossless decomposition of R.
- A JD ⋈ {R1, R2,..., Rn} is said to hold over a relation R if R1, R2,....., Rn is a lossless-join decomposition.
- The *(A, B, C, D), (C, D) will be a JD of R if the join of join's attribute is equal to the relation R.

Here, *(R1, R2, R3) is used to indicate that relation R1, R2, R3 and so on are a JD of R.

Fifth normal form (5NF)

- A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.
- 5NF is satisfied when all the tables are broken into as many tables as possible in order to avoid redundancy.
- 5NF is also known as Project-join normal form (PJ/NF).

Example

SUBJECT	LECTURER	SEMESTER
Computer	Anshika	Semester 1
Computer	John	Semester 1
Math	John	Semester 1
Math	Akash	Semester 2
Chemistry	Praveen	Semester 1

In the above table, John takes both Computer and Math class for Semester 1 but he doesn't take Math class for Semester 2. In this case, combination of all these fields required to identify a valid data.

Suppose we add a new Semester as Semester 3 but do not know about the subject and who will be taking that subject so we leave Lecturer and Subject as NULL. But all three columns together acts as a primary key, so we can't leave other two columns blank.

So to make the above table into 5NF, we can decompose it into three relations P1, P2 & P3:

P1

SEMESTER	SUBJECT
Semester 1	Computer
Semester 1	Math
Semester 1	Chemistry
Semester 2	Math

P2

SUBJECT	LECTURER
Computer	Anshika
Computer	John
Math	John
Math	Akash
Chemistry	Praveen

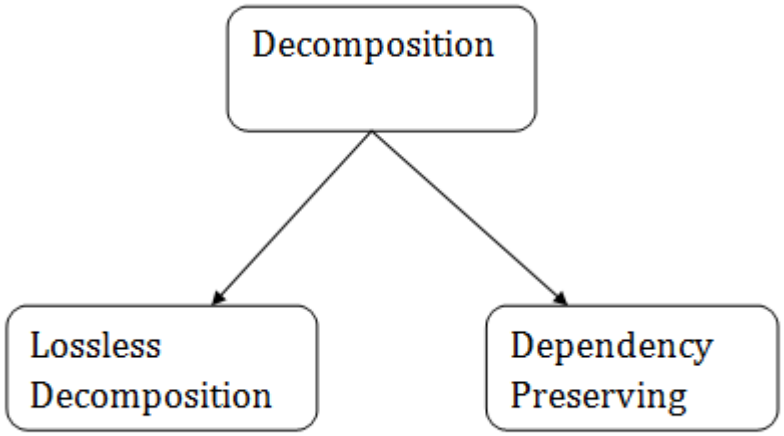
P3

SEMSTER	LECTURER
Semester 1	Anshika
Semester 1	John
Semester 1	John
Semester 2	Akash
Semester 1	Praveen

Relational Decomposition

- When a relation in the relational model is not in appropriate normal form then the decomposition of a relation is required.
- In a database, it breaks the table into multiple tables.
- If the relation has no proper decomposition, then it may lead to problems like loss of information.
- Decomposition is used to eliminate some of the problems of bad design like anomalies, inconsistencies, and redundancy.

Types of Decomposition



Lossless Decomposition

- If the information is not lost from the relation that is decomposed, then the decomposition will be lossless.
- The lossless decomposition guarantees that the join of relations will result in the same relation as it was decomposed.
- The relation is said to be lossless decomposition if natural joins of all the decomposition give the original relation.

Example:

EMPLOYEE_DEPARTMENT table:

EMP_ID	EMP_NAME	EMP_AGE	EMP_CITY	DEPT_ID	DEPT_NAME
22	Denim	28	Mumbai	827	Sales
33	Alina	25	Delhi	438	Marketing
46	Stephan	30	Bangalore	869	Finance
52	Katherine	36	Mumbai	575	Production
60	Jack	40	Noida	678	Testing

The above relation is decomposed into two relations EMPLOYEE and DEPARTMENT

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_AGE	EMP_CITY
22	Denim	28	Mumbai
33	Alina	25	Delhi

46	Stephan	30	Bangalore
52	Katherine	36	Mumbai
60	Jack	40	Noida

DEPARTMENT table

DEPT_ID	EMP_ID	DEPT_NAME
827	22	Sales
438	33	Marketing
869	46	Finance
575	52	Production
678	60	Testing

Now, when these two relations are joined on the common column "EMP_ID", then the resultant relation will look like:

Employee ⋈ Department

EMP_ID	EMP_NAME	EMP_AGE	EMP_CITY	DEPT_ID	DEPT_NAME
22	Denim	28	Mumbai	827	Sales
33	Alina	25	Delhi	438	Marketing
46	Stephan	30	Bangalore	869	Finance
52	Katherine	36	Mumbai	575	Production
60	Jack	40	Noida	678	Testing

Hence, the decomposition is Lossless join decomposition.

Dependency Preserving

- It is an important constraint of the database.
- In the dependency preservation, at least one decomposed table must satisfy every dependency.
- If a relation R is decomposed into relation R1 and R2, then the dependencies of R either must be a part of R1 or R2 or must be derivable from the combination of functional dependencies of R1 and R2.
- For example, suppose there is a relation R (A, B, C, D) with functional dependency set (A->BC). The relational R is decomposed into R1(ABC) and R2(AD) which is dependency preserving because FD A->BC is a part of relation R1(ABC).

Query Processing and Optimization:

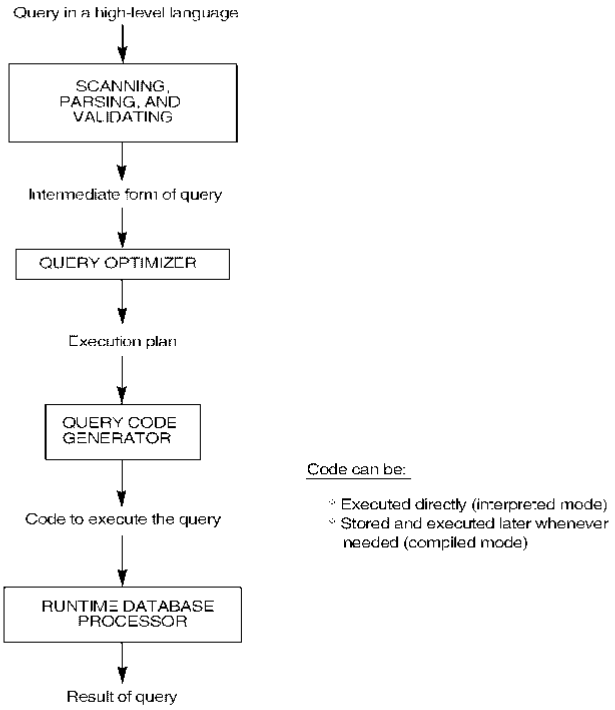
Query: A query is a request for information from a database.

Query block: the basic unit that can be translated into the algebraic operators and optimized.

A query block contains a single SELECT-FROM-WHERE expression, as well as GROUP BY and HAVING clause if these are part of the block.

Query Plans: A query plan (or query execution plan) is an ordered set of steps used to access data in a SQL relational database management system.

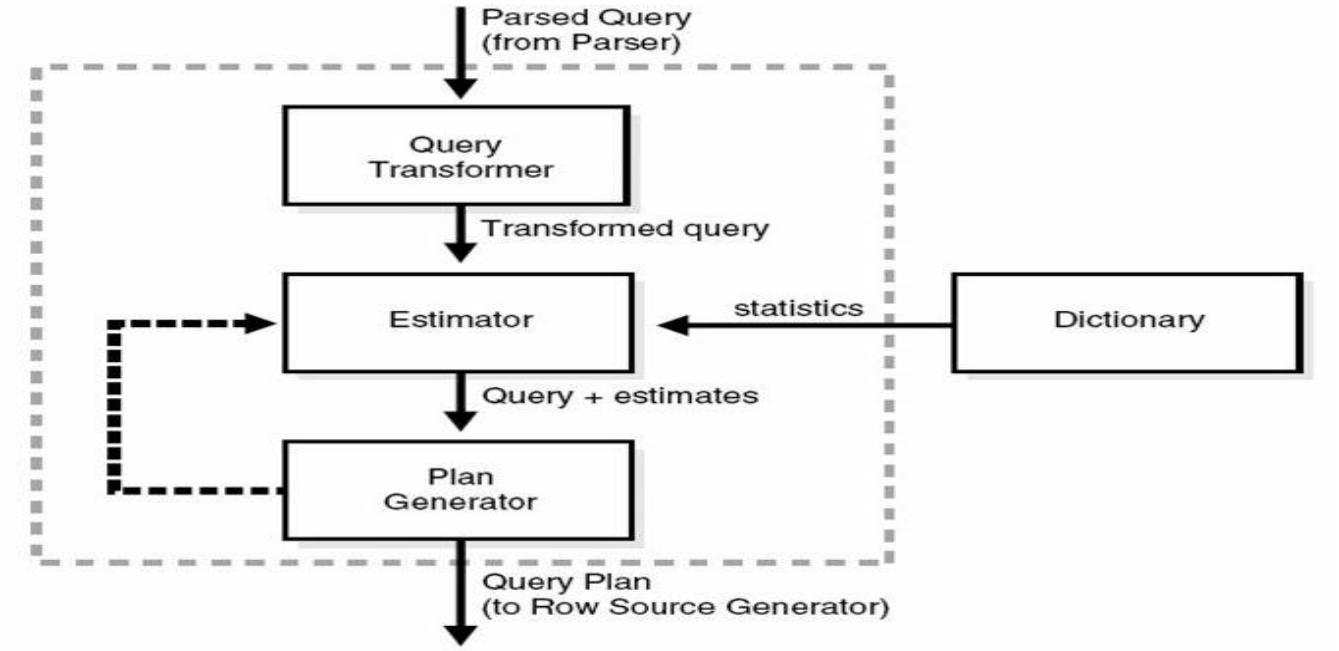
Figure 18.1 Typical steps when processing a high-level query.



Query Optimization: A single query can be executed through different algorithms or re-written in different forms and structures. Hence, the question of query optimization comes into the picture – Which of these forms or pathways is the most optimal? The query optimizer attempts to determine the most efficient way to execute a given query by considering the possible query plans.

Importance: The goal of query optimization is to reduce the system resources required to fulfil a query, and ultimately provide the user with the correct result set faster.

- Query optimization provides faster query processing.
- It requires less cost per query.
- It gives less stress to the database.
- It provides high performance of the system.
- It consumes less memory.



Steps for Query Optimization

Query optimization involves three steps, namely query tree generation, plan generation, and query plan code generation.

Step 1 − Query Tree Generation

A query tree is a tree data structure representing a relational algebra expression. The tables of the query are represented as leaf nodes. The relational algebra operations are represented as the internal nodes. The root represents the query as a whole.

During execution, an internal node is executed whenever its operand tables are available. The node is then replaced by the result table. This process continues for all internal nodes until the root node is executed and replaced by the result table.

For example, let us consider the following schemas −

EMPLOYEE

EmpID EName Salary DeptNo DateOfJoining

DEPARTMENT

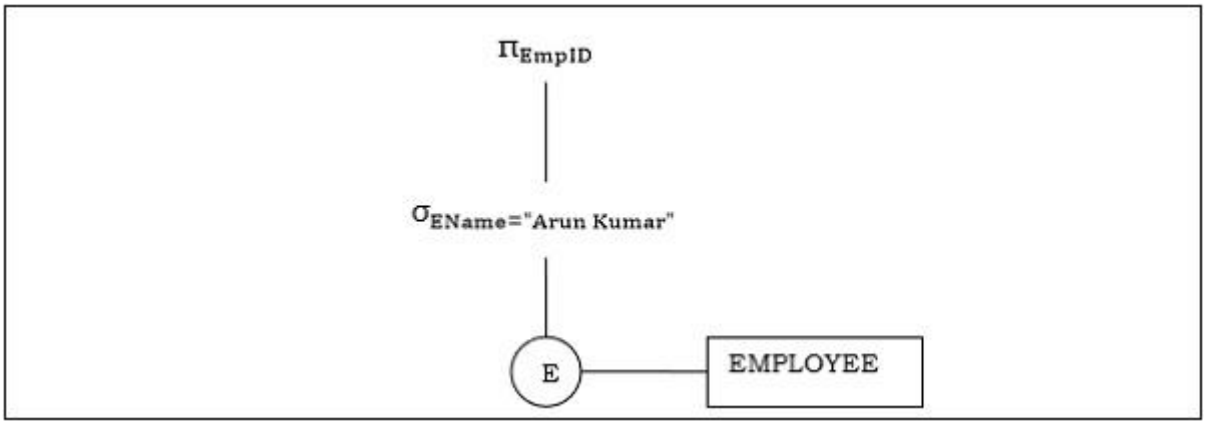
DNo DName Location

Example 1

Let us consider the query as the following.

$$\pi_{EmpID}(\sigma_{EName="ArunKumar"}(EMPLOYEE))$$

The corresponding query tree will be −

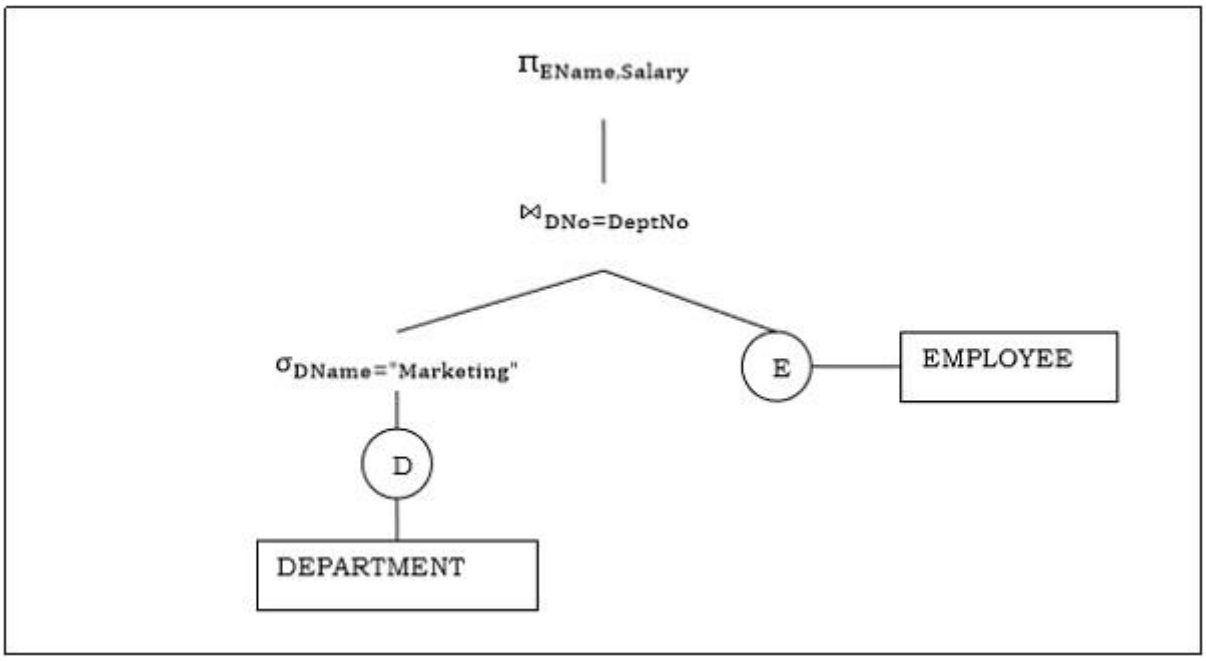


Example 2

Let us consider another query involving a join.

πEName,Salary(*σDName="Marketing"*(*DEPARTMENT*))⋈*DNo=DeptNo*(*EMPLOYEE*)

Following is the query tree for the above query.



Step 2 − Query Plan Generation

After the query tree is generated, a query plan is made. A query plan is an extended query tree that includes access paths for all operations in the query tree. Access paths specify how the relational operations in the tree should be performed. For example, a selection operation can have an access path that gives details about the use of B+ tree index for selection.

Besides, a query plan also states how the intermediate tables should be passed from one operator to the next, how temporary tables should be used and how operations should be pipelined/combined.

Step 3− Code Generation

Code generation is the final step in query optimization. It is the executable form of the query, whose form depends upon the type of the underlying operating system. Once the query code is generated, the Execution Manager runs it and produces the results.

Approaches to Query Optimization

Among the approaches for query optimization, exhaustive search and heuristics-based algorithms are mostly used.

Exhaustive Search Optimization

In these techniques, for a query, all possible query plans are initially generated and then the best plan is selected. Though these techniques provide the best solution, it has an exponential time and space complexity owing to the large solution space. For example, dynamic programming technique.

Heuristic Based Optimization

Heuristic based optimization uses rule-based optimization approaches for query optimization. These algorithms have polynomial time and space complexity, which is lower than the exponential complexity of exhaustive search-based algorithms. However, these algorithms do not necessarily produce the best query plan.

Some of the common heuristic rules are −

- Perform select and project operations before join operations. This is done by moving the select and project operations down the query tree. This reduces the number of tuples available for join.
- Perform the most restrictive select/project operations at first before the other operations.
- Avoid cross-product operation since they result in very large-sized intermediate tables.

Cost Estimation: The main of Query Optimization is to choose the most efficient way of implement the relation algebra operations at the lowest cost.

Query Optimization should not dependent solely, but it should also estimate the cost of executing the different strategies and find the strategies with minimum cost estimate.

The cost function is used in query optimization is estimates and but exact cost function

The cost of an operation is heavily dependent on its selectivity that is proportion of select operation that forms output.

Different algorithms are suitable for low or high selectivity queries

Cost of algorithm is dependent on cardinality.

1. Access cost to secondary storage
2. Storage cost
3. Computation cost
4. Memory usage cost
5. Communication cost