CHANDIGARH ENGINEERING COLLEGE

B.TECH.(CSE 5th Sem.)

Notes Subject: Database Management System

Subject Code: ((BTCS 501-18))

Unit 2

Relational Query Language in DBMS

SQL has its own querying methods to interact with the database. But how do these queries work in the database? These queries work similarly to Relational Algebra that we study in mathematics. In the database, we have tables participating in relational Algebra.

Relational Database systems are expected to be equipped with a query language that assists users to query the database. Relational Query Language is used by the user to communicate with the database user requests for the information from the database. Relational algebra breaks the user requests and instructs the DBMS to execute the requests. It is the language by which the user communicates with the database. They are generally on a higher level than any other programming language. These relational query languages can be Procedural and Non-Procedural.

Types of Relational Query Language

There are two types of relational query language:

- Procedural Query Language
- Non-Procedural Language

Procedural Query Language

In Procedural Language, the user instructs the system to perform a series of operations on the database to produce the desired results. Users tell what data to be retrieved from the database and how to retrieve it. Procedural Query Language performs a set of queries instructing the DBMS to perform various transactions in sequence to meet user requests.

Relational Algebra is a Procedural Query Language

Relational Algebra could be defined as the set of operations on relations.

There are a few operators that are used in relational algebra -

1. Select (sigma): Returns rows of the input relation that satisfy the provided predicate. It is unary Operator means requires only one operand.
2. Projection (π): Show the list of those attribute which we desire to appear and rest other attributes are eliminated from the table. It seperates the table vertically.
3. Set Difference (-): It returns the difference between two relations . If we have two relations R and S them R-S will return all the tuples (row) which are in relation R but not in Relation S , It is binary operator.
4. Cartesian Product (X): Combines every tuple (row) of one table with every tuple (row) in other table ,also referred as cross Product . It is a binary operator.
5. Union (U): Outputs the union of tuples from both the relations. Duplicate tuples are eliminated automatically. It is a binary operator means it require two operands.

Non-Procedural Language

In Non Procedural Language user outlines the desired information without giving a specific procedure or without telling the steps by step process for attaining the information. It only gives a single Query on one or more tables to get .The user tells what is to be retrieved from the database but does not tell how to accomplish it.

For Example: get the name and the contact number of the student with a Particular ID will have a single query on STUDENT table.

Relational Calculus is a Non Procedural Language .

Relational Calculus exists in two forms:

1. Tuple Relational Calculus (TRC): Tuple Relational Calculus is a non procedural query language , It is used for selecting the tuples that satisfy the given condition or predicate . The result of the relation can have one or more tuples (row).
2. Domain Relational Calculus (DRC): Domain Relational Calculus is a Non Procedural Query Language , the records are filtered based on the domains , DRC uses the list of attributes to be selected from relational based on the condition.

Introduction of Relational Algebra in DBMS

Relational algebra consists of a certain set of rules or operations that are widely used to manipulate and query data from a relational database. It can be facilitated by utilizing SQL language and helps users interact with database tables based on querying data from the database more efficiently and effectively.

Relational Algebra incorporates a collection of operations, such as filtering data or combining data, that help us organize and manipulate data more efficiently. This ” algebra ” is the foundation for most database queries, and it enables us to extract the required information from the databases by using SQL query language.

Fundamental Operators

Relational algebra consists of various operators that help us fetch and manipulate data from relational tables in the database to perform certain operations on relational data.

These are the basic/fundamental operators used in Relational Algebra

1. Selection(σ)
2. Projection(π)
3. Union(U)
4. Set Difference(-)
5. Set Intersection(∩)
6. Rename(ρ)
7. Cartesian Product(X)

1. Selection(σ) : Selection Operation is basically used to filter out rows from a given table based on certain given condition. It basically allows you to retrieve only those rows that match the condition as per condition passed during SQL Query.

It is used to select required tuples of the relations.

Illustration : If we want to get the details of all the work in the Department “IT”, we would use the selection operator to filter out these based on the given condition.

Example:

A	B	C
1	2	4
2	2	3
3	2	3
4	3	4

Output Table :

For the above relation, **σ(c>3)R** will select the tuples which have c more than 3.

A	B	C
1	2	4
4	3	4

Note: The selection operator only selects the required tuples but does not display them. For display, the data projection operator is used.

2. Projection(π) : While Selection operation works on rows , similarly projection operation of relational algebra works on columns. It basically allows you to pick specific columns from a given relational table based on the given condition and ignoring all the other remaining columns.

Illustration : If we are interested in the specific columns from the relational tables, we would prefer to use the Projection Operator.

It is used to project required column data from a relation.

Example: Consider Table 1. Suppose we want columns B and C from Relation R.

π(B,C)R will show following columns.

B	C
2	4
2	3
3	4

Note: By Default, projection removes duplicate data.

3. Union(U) : Union Operator is basically used to combine the results of two queries into a single result. The only condition is that both queries must return same number of columns with same data types.

Illustration : If in case we want a list of all the employee from two different department . Then in that case we should use union operation to merge both the list from two different department.

Union operation in relational algebra is the same as union operation in set theory.

Example: FRENCH

Student_Name	Roll_Number
Ram	01
Mohan	02
Vivek	13
Geeta	17

GERMAN

Student_Name	Roll_Number
Vivek	13
Geeta	17
Shyam	21

Student_Name	Roll_Number
Rohan	25

Consider the following table of Students having different optional subjects in their course.

π(Student_Name)FRENCH U π(Student_Name)GERMAN

Student_Name
Ram
Mohan
Vivek
Geeta
Shyam
Rohan

Note: The only constraint in the union of two relations is that both relations must have the same set of Attributes.

4. Set Difference(-) : Set difference basically provides the rows that are present in one table , but not in another tables.

Illustration : If you have two lists of employees, one from the IT Department and one from the Marketing department & you want to know which employees are in IT but not in Marketing , the set difference operator would used.

Set Difference in relational algebra is the same set difference operation as in set theory.

Example: From the above table of FRENCH and GERMAN, Set Difference is used as follows

π(Student_Name)FRENCH - π(Student_Name)GERMAN

Student_Name
Ram

Student_Name
Mohan

Note: The only constraint in the Set Difference between two relations is that both relations must have the same set of Attributes.

5. Set Intersection(∩) : Set Intersection basically allows to fetches only those rows of data that are common between two sets of relational tables.

Illustration : If we want to find the number of employees who work in both IT and Marketing Department , then in that case we use Intersection Operator.

Set Intersection in relational algebra is the same set intersection operation in set theory.

Example: From the above table of FRENCH and GERMAN, the Set Intersection is used as follows:

π(Student_Name)FRENCH ∩ π(Student_Name)GERMAN

Student_Name
Vivek
Geeta

Relational Calculus

As Relational Algebra is a procedural query language, Relational Calculus is a non-procedural query language. It basically deals with the end results. It always tells me what to do but never tells me how to do it.

There are two types of Relational Calculus

1. Tuple Relational Calculus(TRC)
2. Domain Relational Calculus(DRC)
3. **Tuple Relational Calculus (TRC)** is a non-procedural query language used in relational database management systems (RDBMS) to retrieve data from tables. TRC is based on the concept of tuples, which are ordered sets of attribute values that represent a single row or record in a database table.
4. TRC is a declarative language, meaning that it specifies what data is required from the database, rather than how to retrieve it. TRC queries are expressed as logical formulas that describe the desired tuples.
5. **Syntax:** The basic syntax of TRC is as follows:
6. { t | P(t) }
7. where t is a **tuple variable** and P(t) is a **logical formula** that describes the conditions that the tuples in the result must satisfy. The **curly braces {}** are used to indicate that the expression is a set of tuples.

8. For example, let's say we have a table called "Employees" with the following attributes:

Employee ID
Name
Salary
Department ID

Domain Relational Calculus is a non-procedural query language equivalent in power to Tuple Relational Calculus. Domain Relational Calculus provides only the description of the query but it does not provide the methods to solve it. In Domain Relational Calculus, a query is expressed as,

{ < x1, x2, x3, ..., xn > | P (x1, x2, x3, ..., xn) }

where, < x1, x2, x3, …, xn> represents resulting domains variables and P (x1, x2, x3, …, xn) represents the condition or formula equivalent to the Predicate calculus.

Predicate Calculus Formula:

1. Set of all comparison operators
2. Set of connectives like and, or, not
3. Set of quantifiers

Example:

Table-1: Customer

Customer name	**Street**	**City**
Debomit	Kadamtala	Alipurduar
Sayantan	Udaypur	Balurghat
Soumya	Nutanchati	Bankura
Ritu	Juhu	Mumbai

Table-2: Loan

Loan number	Branch name	Amount
L01	Main	200
L03	Main	150
L10	Sub	90
L08	Main	60

Table-3: Borrower

Customer name	Loan number
Ritu	L01
Debomit	L08
Soumya	L03

Query-1: Find the loan number, branch, amount of loans of greater than or equal to 100 amount.

$\{\prec l, b, a\succ \mid \prec l, b, a\succ \in loan \wedge (a \geq 100)\}$

Resulting relation:

Loan number	Branch name	Amount
L01	Main	200
L03	Main	150

Query-2: Find the loan number for each loan of an amount greater or equal to 150.

$\{\prec l\succ \mid \exists\ b, a\ (\prec l, b, a\succ \in loan \wedge (a \geq 150))\}$

Resulting relation:

Open Source Database?

An open-source database is a database where anyone can easily view the source code and this is open and free to download. Also for the community version, some small additional and affordable costs are

imposed. Open Source Database provides Limited technical support to end-users. Here Installation and updates are administered by the user.

Advantages of Open Source Databases

- **Cost:** Open-source databases are generally free, which means they can be used without any licensing fees.
- **Customization:** Since the source code is available, developers can modify and customize the database to meet specific requirements.
- **Community Support:** Open-source databases have a large community of users who contribute to documentation, bug fixes, and improvements.
- **Security:** With open-source databases, security vulnerabilities can be detected and fixed quickly by the community.
- **Scalability:** Open-source databases are typically designed to be scalable, which means they can handle large amounts of data and traffic.

Disadvantages of Open Source Databases

- **Limited Technical Support:** While there is a large community of users who can help troubleshoot issues, there is no guarantee of professional technical support.
- **Complexity:** Open source databases can be more difficult to set up and configure than commercial databases, especially for users who are not experienced in database administration.
- **Lack of Features:** Open source databases may not have all the features that are available in commercial databases, such as advanced analytics and reporting tools.

What is Commercial Database?

Commercial databases are those that have been created for Commercial Purposes only. They are premium and are not free like Open Source Database. In Commercial Database it is guaranteed that technical support is provided. In this Installation, updates are Administered by the Software Vendor. **For example:** Oracle, IBM DB2, etc.

Advantages of Commercial Databases

- **Technical Support:** Commercial databases usually come with professional technical support, which can be helpful for organizations that need assistance with setup, configuration, or troubleshooting.
- **Features:** Commercial databases typically have more features than open-source databases, including advanced analytics, reporting, and data visualization tools.
- **Security:** Commercial databases often have built-in security features and can provide better protection against cyber threats.
- **Integration:** Commercial databases are often designed to work seamlessly with other enterprise software, making integration with existing systems easier.

Disadvantages of Commercial Databases

- **Cost:** Commercial databases can be expensive, with licensing fees and maintenance costs that can add up over time.

- **Vendor Lock-In:** Organizations that use commercial databases may become dependent on the vendor and find it difficult to switch to another database.
- **Limited Customization:** Commercial databases may not be as customizable as open source databases, which can be a disadvantage for organizations with specific requirements.

Loan number
L01
L03

Query-3: Find the names of all customers having a loan at the "Main" branch and find the loan amount .

{≺c, a≻ | ∃ l (≺c, l≻ ∈ borrower ∧ ∃ b (≺l, b, a≻ ∈ loan ∧ (b = "Main")))}

Resulting relation:

Customer Name	Amount
Ritu	200
Debomit	60
Soumya	150

MySQL is a popular open-source **Relational Database Management System** (RDBMS) that uses **SQL** (Structured Query Language) for database operations. While MySQL is a specific database system accessible for free and supports various programming languages.

In this article, we will explore the importance of **MySQL** with its uses and discover why it is so important in databases.

MySQL is a popular choice for managing **relational databases** for several reasons:

1. **Open Source:** MySQL is open-source software, which means it's **free to use** and has a large community of developers contributing to its improvement.
2. **Relational:** MySQL follows the relational database model, allowing users to organize data into **tables** with **rows** and **columns**, facilitating efficient **data storage** and retrieval.
3. **Reliability:** MySQL has been around for a long time and is known for its **stability** and **reliability**.

4. **Performance:** MySQL is optimized for performance, making it capable of handling **high-volume transactions** and large datasets efficiently.
5. **Scalability:** MySQL can scale both **vertically** and **horizontally** to accommodate growing data and user loads. You can add more resources to a single server or distribute the workload across multiple servers using techniques like sharding or replication.
6. **Compatibility:** MySQL is widely supported by many **programming languages**, frameworks, and tools. It offers connectors and APIs for popular languages like PHP, Python, Java, and more, making it easy to integrate with your existing software stack.
7. **Security:** MySQL provides robust **security features** to protect your data, including access controls, encryption, and auditing capabilities. With proper configuration, you can ensure that only authorized users have access to sensitive information.

Data dependency in a database management system (DBMS) is the relationship between the data stored in tables. Here are some types of data dependencies in DBMS:

- **Transitive dependency**

When one attribute's value determines another's value through a third attribute.

- **Partial dependency**

When one primary key determines some other attribute or attributes.

- **Trivial functional dependency**

When an attribute or a set of attributes uniquely determines itself or a part of itself. For example, if A leads to A or A, B leads to A.

- **Multivalued dependency**

When there are multiple rows in a given table. Any multivalued dependency would at least involve three attributes of any table.

- **Full dependency**

When the determinant of the dependency is either a candidate key or a super key.

Domain dependency is another type of dependency in DBMS, which is the relationship between two or more tables in the database.

Data dependence also refers to the extent to which an application program is exposed to changes made to an external source, such as a database or file.

Armstrong Axioms

The term Armstrong Axioms refers to the sound and complete set of inference rules or axioms, introduced by William W. Armstrong, that is used to test the logical implication of **functional dependencies**. If F is a set of functional dependencies then the closure of F, denoted as F+, is the set of all functional dependencies logically implied by F. Armstrong's Axioms are a set of rules, that when applied repeatedly, generates a closure of functional dependencies.

Axioms

- **Axiom of Reflexivity:** If A is a set of attributes and B is a subset of A, then A holds B. If $B \subseteq A$ then $A \rightarrow B$. This property is trivial property.

- **Axiom of Augmentation:** If **A→B** holds and Y is the attribute set, then **AY→BY** also holds. That is adding attributes to dependencies, does not change the basic dependencies. If **A→B**, then **AC→BC** for any C.
- **Axiom of Transitivity:** Same as the transitive rule in algebra, if **A→B** holds and **B→C** holds, then **A→C** also holds. **A→B** is called A functionally which determines B. If **X→Y** and **Y→Z**, then **X→Z.**

Secondary Rules

These rules can be derived from the above axioms.

- **Union:** If **A→B** holds and **A→C** holds, then **A→BC** holds. If **X→Y** and **X→Z** then **X→YZ**.
- **Composition:** If **A→B** and **X→Y** hold, then **AX→BY** holds.
- **Decomposition:** If **A→BC** holds then **A→B** and **A→C** hold. If **X→YZ** then **X→Y** and **X→Z**.
- **Pseudo Transitivity:** If **A→B** holds and **BC→D** holds, then **AC→D** holds. If **X→Y** and **YZ→W** then **XZ→W**.
- **Self Determination:** It is similar to the Axiom of Reflexivity, i.e. **A→A** for any A.
- **Extensivity:** Extensivity is a case of augmentation. If **AC→A,** and **A→B**, then **AC→B**. Similarly, **AC→ABC** and **ABC→BC**. This leads to **AC→BC**.

Armstrong Relation

Armstrong Relation can be stated as a relation that is able to satisfy all functional dependencies in the F+ Closure. In the given set of dependencies, the size of the minimum Armstrong Relation is an exponential function of the number of attributes present in the dependency under consideration.

Normal Forms in DBMS

Normalization is the process of minimizing **redundancy** from a relation or set of relations. Redundancy in relation may cause insertion, deletion, and update anomalies. So, it helps to minimize the redundancy in relations. **Normal forms** are used to eliminate or reduce redundancy in database tables.

Normalization of DBMS

In database management systems (DBMS), normal forms are a series of guidelines that help to ensure that the design of a database is efficient, organized, and free from data anomalies. There are several levels of normalization, each with its own set of guidelines, known as normal forms.

Important Points Regarding Normal Forms in DBMS

- **First Normal Form (1NF):** This is the most basic level of normalization. In 1NF, each table cell should contain only a single value, and each column should have a unique name. The first normal form helps to eliminate duplicate data and simplify queries.
- **Second Normal Form (2NF):** 2NF eliminates redundant data by requiring that each non-key attribute be dependent on the primary key. This means that each column should be directly related to the primary key, and not to other columns.
- **Third Normal Form (3NF):** 3NF builds on 2NF by requiring that all non-key attributes are independent of each other. This means that each column should be directly related to the primary key, and not to any other columns in the same table.

- **Boyce-Codd Normal Form (BCNF):** BCNF is a stricter form of 3NF that ensures that each determinant in a table is a candidate key. In other words, BCNF ensures that each non-key attribute is dependent only on the candidate key.
- **Fourth Normal Form (4NF):** 4NF is a further refinement of BCNF that ensures that a table does not contain any multi-valued dependencies.
- **Fifth Normal Form (5NF):** 5NF is the highest level of normalization and involves decomposing a table into smaller tables to remove data redundancy and improve data integrity.

Normal forms help to reduce data redundancy, increase data consistency, and improve database performance. However, higher levels of normalization can lead to more complex database designs and queries. It is important to strike a balance between normalization and practicality when designing a database.

Advantages of Normal Form

- **Reduced data redundancy:** Normalization helps to eliminate duplicate data in tables, reducing the amount of storage space needed and improving database efficiency.
- **Improved data consistency:** Normalization ensures that data is stored in a consistent and organized manner, reducing the risk of data inconsistencies and errors.
- **Simplified database design:** Normalization provides guidelines for organizing tables and data relationships, making it easier to design and maintain a database.
- **Improved query performance:** Normalized tables are typically easier to search and retrieve data from, resulting in faster query performance.
- **Easier database maintenance:** Normalization reduces the complexity of a database by breaking it down into smaller, more manageable tables, making it easier to add, modify, and delete data.

Overall, using normal forms in DBMS helps to improve data quality, increase database efficiency, and simplify database design and maintenance.

First Normal Form

If a relation contain composite or multi-valued attribute, it violates first normal form or a relation is in first normal form if it does not contain any composite or multi-valued attribute. A relation is in first normal form if every attribute in that relation is **singled valued attribute**.

- **Example 1 –** Relation STUDENT in table 1 is not in 1NF because of multi-valued attribute STUD_PHONE. Its decomposition into 1NF has been shown in table 2.

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_C
1	RAM	9716271721, 9871717178	HARYANA	INDIA
2	RAM	9898297281	PUNJAB	INDIA
3	SURESH		PUNJAB	INDIA

Table 1

Conversion to first normal form

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_C
1	RAM	9716271721	HARYANA	INDIA
1	RAM	9871717178	HARYANA	INDIA
2	RAM	9898297281	PUNJAB	INDIA
3	SURESH		PUNJAB	INDIA

Table 2

Example

- **Example 2 –**

ID Name Courses
---
1 A c1, c2
2 E c3
3 M C2, c3

- In the above table Course is a multi-valued attribute so it is not in 1NF. Below Table is in 1NF as there is no multi-valued attribute

ID Name Course
---
1 A c1
1 A c2
2 E c3
3 M c2
3 M c3

Second Normal Form

A relation is in 2NF if it is in 1NF and any non-prime attribute (attributes which are not part of any candidate key) is not partially dependent on any proper subset of any candidate key of the table. In other words, we can say that, every non-prime attribute must be fully dependent on each candidate key.

A functional dependency X->Y (where X and Y are set of attributes) is said to be in **partial dependency**, if Y can be determined by any proper subset of X.

However, in 2NF it is possible for a prime attribute to be partially dependent on any candidate key, but every non-prime attribute must be fully dependent(or not partially dependent) on each candidate key of the table.

- **Example 1 –** Consider table-3 as following below.

STUD_NO	COURSE_NO	COURSE_FEE
1	C1	1000
2	C2	1500
1	C4	2000
4	C3	1000
4	C1	1000
2	C5	2000

- {Note that, there are many courses having the same course fee} Here, COURSE_FEE cannot alone decide the value of COURSE_NO or STUD_NO; COURSE_FEE together with STUD_NO cannot decide the value of COURSE_NO; COURSE_FEE together with COURSE_NO cannot decide the value of STUD_NO; Hence, COURSE_FEE would be a non-prime attribute, as it does not belong to the one only candidate key {STUD_NO, COURSE_NO} ; But, COURSE_NO -> COURSE_FEE, i.e., COURSE_FEE is dependent on COURSE_NO, which is a proper subset of the candidate key. Non-prime attribute COURSE_FEE is dependent on a proper subset of the candidate key, which is a partial dependency and so this relation is not in 2NF. To convert the above relation to 2NF, we need to split the table into two tables such as : Table 1: STUD_NO, COURSE_NO Table 2: COURSE_NO, COURSE_FEE

Table 1

STUD_NO	COURSE_NO
1	C1
2	C2
1	C4
4	C3
4	C1
2	C5

Table 2

COURSE_NO	COURSE_FEE
C1	1000
C2	1500
C3	1000
C4	2000
C5	2000

- **NOTE:** 2NF tries to reduce the redundant data getting stored in memory. For instance, if there are 100 students taking C1 course, we don't need to store its Fee as 1000 for all the 100 records, instead, once we can store it in the second table as the course fee for C1 is 1000.

- **Example 2 –** Consider following functional dependencies in relation R (A, B , C, D)

AB -> C [A and B together determine C]
BC -> D [B and C together determine D]

In the above relation, AB is the only candidate key and there is no partial dependency, i.e., any proper subset of AB doesn't determine any non-prime attribute.

X is a super key.
Y is a prime attribute (each element of Y is part of some candidate key).

Example 1: In relation STUDENT given in Table 4, FD set: {STUD_NO -> STUD_NAME, STUD_NO -> STUD_STATE, STUD_STATE -> STUD_COUNTRY, STUD_NO -> STUD_AGE}

Candidate Key: {STUD_NO}

For this relation in table 4, STUD_NO -> STUD_STATE and STUD_STATE -> STUD_COUNTRY are true.

So STUD_COUNTRY is transitively dependent on STUD_NO. It violates the third normal form.

To convert it in third normal form, we will decompose the relation STUDENT (STUD_NO, STUD_NAME, STUD_PHONE, STUD_STATE, STUD_COUNTRY_STUD_AGE) as: STUDENT

(STUD_NO, STUD_NAME, STUD_PHONE, STUD_STATE, STUD_AGE) STATE_COUNTRY (STATE, COUNTRY)

Consider relation R(A, B, C, D, E) A -> BC, CD -> E, B -> D, E -> A All possible candidate keys in above relation are {A, E, CD, BC} All attributes are on right sides of all functional dependencies are prime.

Example 2: Find the highest normal form of a relation R(A,B,C,D,E) with FD set as {BC->D, AC->BE, B->E}

Step 1: As we can see, (AC)+ ={A,C,B,E,D} but none of its subset can determine all attribute of relation, So AC will be candidate key. A or C can't be derived from any other attribute of the relation, so there will be only 1 candidate key {AC}.

Step 2: Prime attributes are those attributes that are part of candidate key {A, C} in this example and others will be non-prime {B, D, E} in this example.

Step 3: The relation R is in 1st normal form as a relational DBMS does not allow multi-valued or composite attribute. The relation is in 2nd normal form because BC->D is in 2nd normal form (BC is not a proper subset of candidate key AC) and AC->BE is in 2nd normal form (AC is candidate key) and B->E is in 2nd normal form (B is not a proper subset of candidate key AC).

The relation is not in 3rd normal form because in BC->D (neither BC is a super key nor D is a prime attribute) and in B->E (neither B is a super key nor E is a prime attribute) but to satisfy 3rd normal for, either LHS of an FD should be super key or RHS should be prime attribute. So the highest normal form of relation will be 2nd Normal form.

For example consider relation R(A, B, C) A -> BC, B -> A and B both are super keys so above relation is in BCNF.

Third Normal Form

A relation is said to be in third normal form, if we did not have any transitive dependency for non-prime attributes. The basic condition with the Third Normal Form is that, the relation must be in Second Normal Form.

Below mentioned is the basic condition that must be hold in the non-trivial functional dependency X -> Y:

- X is a Super Key.
- Y is a Prime Attribute (this means that element of Y is some part of Candidate Key).

For more, refer to Third Normal Form in DBMS.

BCNF

BCNF (Boyce-Codd Normal Form) is just a advanced version of Third Normal Form. Here we have some additional rules than Third Normal Form. The basic condition for any relation to be in BCNF is that it must be in Third Normal Form.

We have to focus on some basic rules that are for BCNF:

1. Table must be in Third Normal Form.
2. In relation X->Y, X must be a superkey in a relation.

For more, refer to BCNF in DBMS.

Fourth Normal Form

Fourth Normal Form contains no non-trivial multivalued dependency except candidate key. The basic condition with Fourth Normal Form is that the relation must be in BCNF.

The basic rules are mentioned below.

1. It must be in BCNF.
2. It does not have any multi-valued dependency.

For more, refer to Fourth Normal Form in DBMS.

Fifth Normal Form

Fifth Normal Form is also called as Projected Normal Form. The basic conditions of Fifth Normal Form is mentioned below.

Relation must be in Fourth Normal Form.
The relation must not be further non loss decomposed.

For more, refer to Fifth Normal Form in DBMS.

Applications of Normal Forms in DBMS

- **Data consistency:** Normal forms ensure that data is consistent and does not contain any redundant information. This helps to prevent inconsistencies and errors in the database.
- **Data redundancy:** Normal forms minimize data redundancy by organizing data into tables that contain only unique data. This reduces the amount of storage space required for the database and makes it easier to manage.
- **Response time:** Normal forms can improve query performance by reducing the number of joins required to retrieve data. This helps to speed up query processing and improve overall system performance.
- **Database maintenance:** Normal forms make it easier to maintain the database by reducing the amount of redundant data that needs to be updated, deleted, or modified. This helps to improve database management and reduce the risk of errors or inconsistencies.
- **Database design:** Normal forms provide guidelines for designing databases that are efficient, flexible, and scalable. This helps to ensure that the database can be easily modified, updated, or expanded as needed.

Some Important Points about Normal Forms

- BCNF is free from redundancy caused by Functional Dependencies.
- If a relation is in BCNF, then 3NF is also satisfied.
- If all attributes of relation are prime attribute, then the relation is always in 3NF.
- A relation in a Relational Database is always and at least in 1NF form.
- Every Binary Relation (a Relation with only 2 attributes) is always in BCNF.
- If a Relation has only singleton candidate keys(i.e. every candidate key consists of only 1 attribute), then the Relation is always in 2NF(because no Partial functional dependency possible).
- Sometimes going for BCNF form may not preserve functional dependency. In that case go for BCNF only if the lost FD(s) is not required, else normalize till 3NF only.

- There are many more Normal forms that exist after BCNF, like 4NF and more. But in real world database systems it's generally not required to go beyond BCNF.

Query Processing in DBMS

Query Processing is the activity performed in extracting data from the database. In query processing, it takes various steps for fetching the data from the database. The steps involved are:

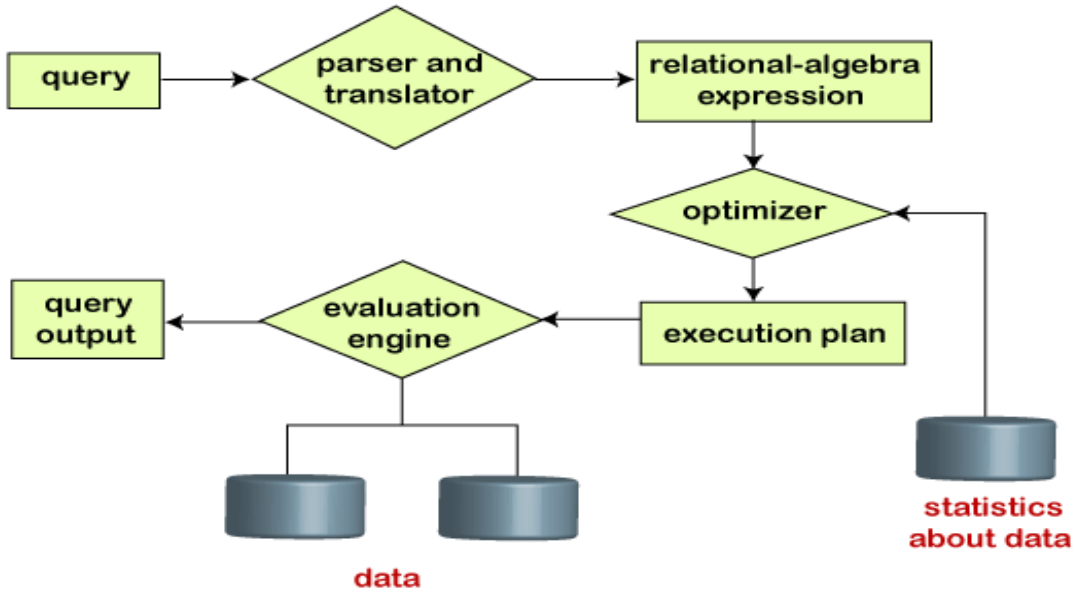
1. Parsing and translation
2. Optimization
3. Evaluation

The query processing works in the following way:

Parsing and Translation

As query processing includes certain activities for data retrieval. Initially, the given user queries get translated in high-level database languages such as SQL. It gets translated into expressions that can be further used at the physical level of the file system. After this, the actual evaluation of the queries and a variety of query -optimizing transformations and takes place. Thus before processing a query, a computer system needs to translate the query into a human-readable and understandable language. Consequently, SQL or Structured Query Language is the best suitable choice for humans. But, it is not perfectly suitable for the internal representation of the query to the system. Relational algebra is well suited for the internal representation of a query. The translation process in query processing is similar to the parser of a query. When a user executes any query, for generating the internal form of the query, the parser in the system checks the syntax of the query, verifies the name of the relation in the database, the tuple, and finally the required attribute value. The parser creates a tree of the query, known as 'parse-tree.' Further, translate it into the form of relational algebra. With this, it evenly replaces all the use of the views when used in the query.

Thus, we can understand the working of a query processing in the below-described diagram:



Steps in query processing

Suppose a user executes a query. As we have learned that there are various methods of extracting the data from the database. In SQL, a user wants to fetch the records of the employees whose salary is greater than or equal to 10000. For doing this, the following query is undertaken:

select emp_name from Employee where salary>10000;

Thus, to make the system understand the user query, it needs to be translated in the form of relational algebra. We can bring this query in the relational algebra form as:

- $\sigma_{salary>10000}$ (π_{salary} (Employee))
- π_{salary} ($\sigma_{salary>10000}$ (Employee))

After translating the given query, we can execute each relational algebra operation by using different algorithms. So, in this way, a query processing begins its working.

Evaluation

For this, with addition to the relational algebra translation, it is required to annotate the translated relational algebra expression with the instructions used for specifying and evaluating each operation. Thus, after translating the user query, the system executes a query evaluation plan.

Query Evaluation Plan

- In order to fully evaluate a query, the system needs to construct a query evaluation plan.
- The annotations in the evaluation plan may refer to the algorithms to be used for the particular index or the specific operations.
- Such relational algebra with annotations is referred to as **Evaluation Primitives**. The evaluation primitives carry the instructions needed for the evaluation of the operation.
- Thus, a query evaluation plan defines a sequence of primitive operations used for evaluating a query. The query evaluation plan is also referred to as **the query execution plan**.
- A **query execution engine** is responsible for generating the output of the given query. It takes the query execution plan, executes it, and finally makes the output for the user query.

Optimization

- The cost of the query evaluation can vary for different types of queries. Although the system is responsible for constructing the evaluation plan, the user does need not to write their query efficiently.
- Usually, a database system generates an efficient query evaluation plan, which minimizes its cost. This type of task performed by the database system and is known as Query Optimization.
- For optimizing a query, the query optimizer should have an estimated cost analysis of each operation. It is because the overall operation cost depends on the memory allocations to several operations, execution costs, and so on.

Joins in DBMS

Join is an operation in DBMS(Database Management System) that combines the row of two or more tables based on related columns between them. The main purpose of Join is to retrieve the data from multiple tables in other words Join is used to perform multi-table queries. It is denoted by ⋈.

Syntax 1

R3 <- ⋈(R1) <join_condition> (R2)

where R1 and R2 are two relations to be joined and R3 is a relation that will hold the result of the join operation.

Example

Temp <- ⋈(student) S.roll==E.roll(Exam)

where S and E are aliases of the student and exam respectively

SQL JOIN Example

Consider the two tables below as follows:

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	HARSH	DELHI	XXXXXXXXXX	18
2	PRATIK	BIHAR	XXXXXXXXXX	19
3	RIYANKA	SILIGURI	XXXXXXXXXX	20
4	DEEP	RAMNAGAR	XXXXXXXXXX	18
5	SAPTARHI	KOLKATA	XXXXXXXXXX	19
6	DHANRAJ	BARABAJAR	XXXXXXXXXX	20
7	ROHIT	BALURGHAT	XXXXXXXXXX	18
8	NIRAJ	ALIPUR	XXXXXXXXXX	19

Table 1 - Student

COURSE_ID	ROLL_NO
1	1
2	2
2	3
3	4
1	5
4	9
5	10
4	11

Table 2 - StudentCourse

Both these tables are connected by one common key (column) i.e. ROLL_NO.

We can perform a JOIN operation using the given SQL query:

SELECT s.roll_no, s.name, s.address, s.phone, s.age, sc.course_id **FROM** Student s **JOIN** StudentCourse sc **ON** s.roll_no = sc.roll_no;

Output:

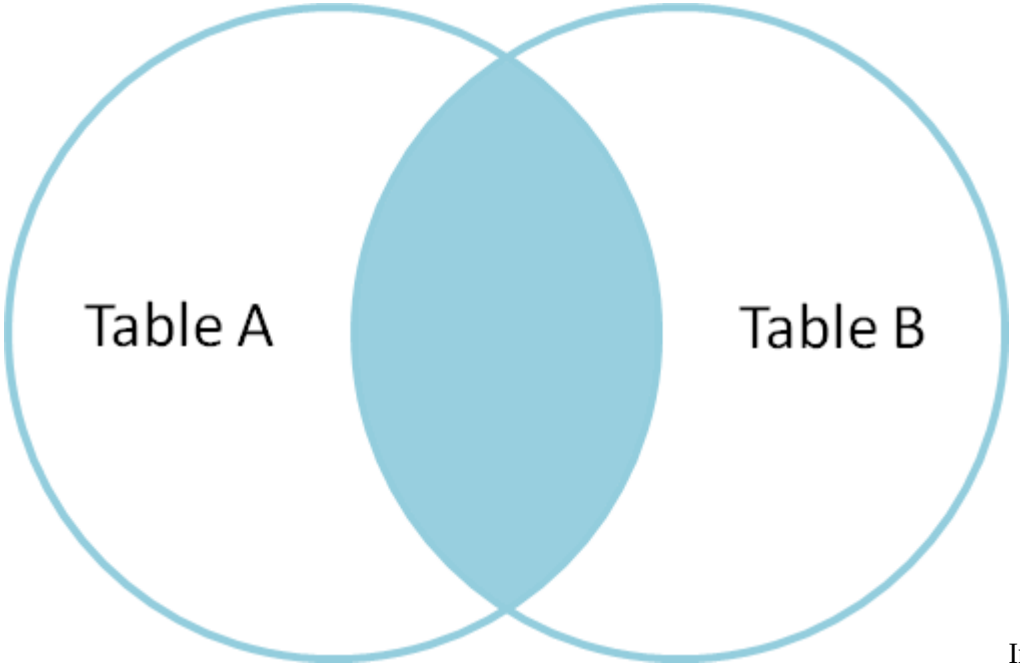
ROLL_NO	NAME	ADDRESS	PHONE	AGE	COURSE_ID
1	HARSH	DELHI	xxxxxxxxxx	18	1
2	PRATIK	BIHAR	xxxxxxxxxx	19	2
3	PRIYANKA	SILIGURI	xxxxxxxxxx	20	2
4	DEEP	RAMNAGAR	xxxxxxxxxx	18	3
5	SAPTARHI	KOLKATA	xxxxxxxxxx	19	1

Types of Join

There are many types of Joins in SQL. Depending on the use case, you can use different types of SQL JOIN clauses. Here are the frequently used SQL JOIN types:

1. Inner Join

Inner Join is a join operation in DBMS that combines two or more tables based on related columns and returns only rows that have matching values among tables. Inner join of two types.



Inner Join

- Conditional join
- Equi Join
- Natural Join

(a) Conditional Join

Conditional join or Theta join is a type of inner join in which tables are combined based on the specified condition.

In conditional join, the join condition can include <, >, <=, >=, ≠ operators in addition to the = operator.

Example: Suppose two tables A and B

Table A

R	S
10	5
7	20

Table B

T	U
10	12
17	6

$A \bowtie S<T\ B$

Output

R	S	T	U
10	5	10	12

SQL Query

SELECT R, S, T, U FROM TableA JOIN TableB ON S < T;

Explanation: This query joins the table A, B and projects attributes R, S, T, U were condition S < T is satisfied.

(b) Equi Join

Equi Join is a type of Inner join in which we use equivalence('=') condition in the join condition

Example: Suppose there are two tables Table A and Table B

Table A

Column A	Column B
a	a
a	b

Table B

Column A	Column B
a	**a**
a	**c**

$A \bowtie A.Column\ B = B.Column\ B\ (B)$

Output

Column A	Column B
a	a

SQL Query

*SELECT * FROM TableA NATURAL JOIN TableB;*

Explanation - The data value "a" is available in both tables Hence we write that "a" is the table in the given output.

(b) Natural Join

Natural join is a type of inner join in which we do not need any comparison operators. In natural join, columns should have the same name and domain. There should be at least one common attribute between the two tables.

Example: Suppose there are two tables Table A and Table B

Table A

Number	**Square**
2	4
3	9

Table B

Number	**Cube**
2	8
3	27

$A \bowtie B$

Output

Number	**Square**	**Cube**
2	4	8

Number	Square	Cube
3	9	27

SQL Query

*SELECT * FROM TableA NATURAL JOIN TableB;*

Explanation - Column Number is available in both tables Hence we write the "Number column once " after combining both tables.

2. Outer Join

Outer join is a type of join that retrieves matching as well as non-matching records from related tables. These three types of outer join

- Left outer join
- Right outer join
- Full outer join

(a) Left Outer Join

It is also called left join. This type of outer join retrieves all records from the left table and retrieves matching records from the right table.

Example: Suppose there are two tables Table A and Table B

Table A

Number	Square
2	4
3	9
4	16

Table B

Number	Cube
2	8
3	27

Number	Cube
5	125

$A \rtimes B$

Output

Number	Square	Cube
2	4	8
3	9	27
4	16	NULL

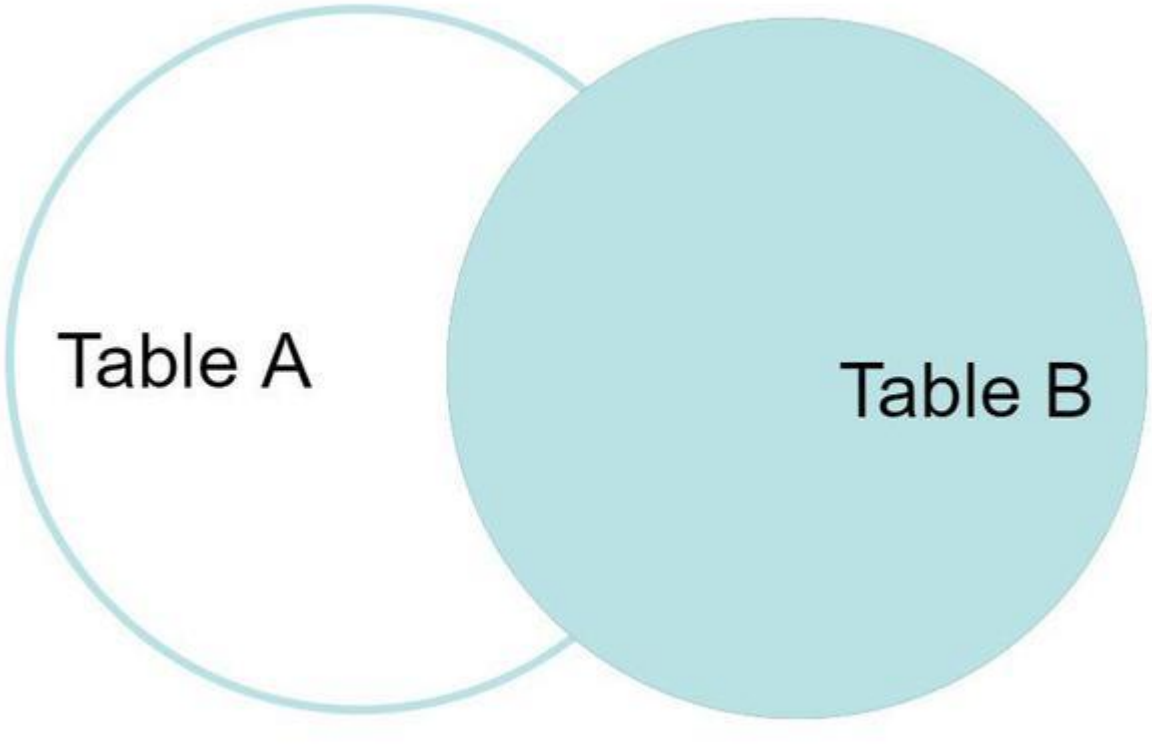
SQL Query-

SELECT * FROM TableA LEFT OUTER JOIN TableB ON TableA.Number = TableB.Number;

Explanation: Since we know in the left outer join we take all the columns from the left table (Here Table A) In the table A we can see that there is no Cube value for number 4. so we mark this as NULL.

(b) Right Outer Join

It is also called a right join. This type of outer join retrieves all records from the right table and retrieves matching records from the left table. And for the record which doesn't lies in Left table will be marked as NULL in result Set.



Right Outer Join

Example: Suppose there are two tables Table A and Table B

$A \rtimes B$

Output:

Number	Square	Cube
2	4	8
3	9	27
5	NULL	125

SQL Query

SELECT * FROM TableA RIGHT OUTER JOIN TableB ON TableA.Number= TableB.Number;

Explanation: Since we know in the right outer join we take all the columns from the right table (Here Table B) In table A we can see that there is no square value for number 5. So we mark this as NULL.

(c) Full Outer Join

FULL JOIN creates the result set by combining the results of both LEFT JOIN and RIGHT JOIN. The result set will contain all the rows from both tables. For the rows for which there is no matching, the result set will contain *NULL* values.

Example: Table A and Table B are the same as in the left outer join

$A \bowtie B$

Output:

Number	Square	Cube
2	4	8
3	9	27
4	16	NULL
5	NULL	125

SQL Query

SELECT * FROM TableA FULL OUTER JOIN TableB ON TableA.Number= TableB.Number;

Explanation: Since we know in full outer join we take all the columns from both tables (Here Table A and Table B) In the table A and Table B we can see that there is no Cube value for number 4 and No Square value for 5 so we mark this as NULL.