BCA204T: DATA BASE MANAGEMENT SYSTEMS

Unit - III

Functional Dependencies and Normalization for Relational Database: Informal Design Guidelines for Relational schemas, Functional Dependencies, Normal Forms Based on Primary Keys., General Definitions of Second and Third Normal Forms Based on Primary Keys., General Definitions of Second and Third Normal Forms, Boyce-Codd Normal Form. Relational Data Model and Relational Algebra: Relational Model Concepts., relational Model Constraints and relational Database Schema, defining Relations, Update Operations on Relations., Basic Relational Algebra Operations, Additional Relational Operations., Examples of queries in the Relational Algebra., Relational Database design Using ER-to-Relational Mapping.

Unit-III

Functional Dependencies and Normalization for Relational Database

Functional Dependency

Functional dependency (FD) is set of constraints between two attributes in a relation. Functional dependency says that if two tuples have same values for attributes A1, A2,..., An then those two tuples must have to have same values for attributes B1, B2, ..., Bn. Functional dependency is represented by arrow sign (→), that is X→Y, where X functionally determines Y. The left hand side attributes determines the values of attributes at right hand side.

Armstrong's Axioms

William W. Armstrong established a set of rules which can be used to Inference the functional dependencies in a relational database:

- **Reflexivity rule:** A→ B is true, if B is subset of A.
- **Augmentation rule:** If A→ B is true, then AC→BC is also true.
- **Transitivity rule:** If A→ B and B→ C, then A →C is implied.

Codd's 12 Rules

Dr Edgar F. Codd was a computer Scientist who invented Relational model for Database management. Based on relational model relational database was created. Codd came up with twelve rules of his own which according to him, a database must obey in order to be a true relational database.

Codd's rule actually define what quality a DBMS requires in order to become a

Relational Database Management System(RDBMS). Till now, there is hardly

any commercial product that follows all the 13 Codd's rules. Even Oracle follows only eight and half (8.5) out of 12. The Codd's 12 rules are as follows

Rule 1: Information rule

All information is to be represented as stored data in cells of tables. Everything in a database must be stored in table formats. This information can be user data or meta-data.

Rule 2: Guaranteed Access rule

Each unique piece of data(atomic value) should be accessible by :

Table Name + primary key(Row) + Attribute(column).

No other means, such as pointers, can be used to access data.

Rule 3: Systematic Treatment of NULL values

NULL may have several meanings, it can mean data is missing, data is not applicable, or no value. It should be handled consistently.

Rule 4: Active online catalog

The structure description of whole database must be stored in an online catalog, i.e. data dictionary, which can be accessed by the authorized users. Users can use the same query language to access the catalog which they use to access the database itself.

Rule 5: Comprehensive data sub-language rule

A database must have a support for a language which has linear syntax which is capable of data definition, data manipulation and transaction management operations. Database can be accessed by means of this language only, either directly or by means of some application.

Rule 6: View updating rule

All view that are theoretically updatable should be updatable by the system.

Rule 7: High-level insert, update and delete rule

There must be Insert, Delete, Update operations at each level of relations. This must not be limited to a single row, it must also support union, intersection and minus operations to yield sets of data records.

Rule 8: Physical data independence

The application should not have any concern about how the data is physically stored. Also, any change in its physical structure must not have any impact on application.

Rule 9: Logical data independence

If there is change in the logical structure (table structures) of the database the user view of data should not change.

Say, if a table is split into two tables, a new view should give result as the join of the two tables. This rule is most difficult to satisfy.

Rule 10: Integrity independence

Integrity constraints must be defined and separated from the application programs. Changing Constraints must be allowed without affecting the applications.

Rule 11: Distribution independence

A database should work properly regardless of its distribution across a network, whether they are distributed or not. This lays foundation of distributed database.

Rule 12: Non-subversion rule

If low-level access is allowed to a system it should not be able to subvert or bypass the integrity rules to change data.

Normalization

Normalization is the process of organizing the attributes and tables of a relational database to minimize data redundancy.

or

Normalization is the process of reorganizing data in a database so that it meets two basic requirements:

a) There is no redundancy of data (all data is stored in only one place).
b) Data dependencies are logical (all related data items are stored together).

Normalization involves refactoring a table into smaller (and less redundant) tables but without losing information; defining foreign keys in the old table referencing the primary keys of the new ones. The objective is to isolate data so that additions, deletions, and modifications of an attribute can be made in just one table and then propagated through the rest of the database using the defined foreign keys.

Needs for Normalization

- Improves database design.
- Ensure minimum redundancy of data.
- It can save storage space and ensure the consistency of your data.
- More flexible database structure.
- Removes anomalies for database activities.

FIRST NORMAL FORM (1NF)

First normal form: A table is in the first normal form if it contains no repeating columns.

Consider the below table, in this example it shows several employees working on several projects. In this company the same employee can work on different projects and at a different hourly rate. Convert this table into 1NF.

Project Code	Project Title	Project Manager	Project Budget	Emplyoee No	Employee Name	Department No	Department Name	Hourly Rate
PC10	MIS	Nisarga	24500	S10001	ARUN	L004	IT	22
PC10	MIS	Nisarga	24500	S10030	LOKESH	L023	Pionix	18
PC10	MIS	Nisarga	24500	S21010	PAVANI	L004	IT	21
PC45	LIS	Eenchara	17400	S10010	BABU	L004	IT	21
PC45	LIS	Eenchara	17400	S10001	ARUN	L004	IT	18
PC45	LIS	Eenchara	17400	S31002	TULASI	L028	Database	25
PC45	LIS	Eenchara	17400	S13210	WILLIAM	L008	Systems	17
PC64	HMS	Prakruthi	12250	S31002	TULASI	L028	Database	23
PC64	HMS	Prakruthi	12250	S21010	PAVANI	L004	IT	17
PC64	HMS	Prakruthi	12250	S10034	BALU	L009	HR	16

UNF:UnNormalized Table

STEPS:

Transform a table of unnormalised data into first normal form (1NF).

The process is as follows:

- Identify repeating attributes.
- Remove these repeating attributes to a new table together with a **copy** of the key from the UNF table. After removing the duplicate data the repeating attributes are easily identified.
- In the previous table the Employee No, Employee Name, Department No, Department Name and Hourly Rate attributes are repeating. These are the repeating attributes and have been to a new table together with a copy of the original key (ie:Project Code).

- A key of Project Code and Employee No has been defined for this new table. This combination is unique for each row in the table.

1NF Tables: Repeating Attributes Removed

Project Code	Project Title	Project Manager	Project Budget
PC10	MIS	Nisarga	24500
PC45	LIS	Eenchara	17400
PC64	HMS	Prakruthi	12250

Project Code	Emplyoee No	Employee Name	Department No	Department Name	Hourly Rate
PC10	S10001	ARUN	L004	IT	22
PC10	S10030	LOKESH	L023	Pionix	18
PC10	S21010	PAVANI	L004	IT	21
PC45	S10010	BABU	L004	IT	21
PC45	S10001	ARUN	L004	IT	18
PC45	S31002	TULASI	L028	Database	25
PC45	S13210	WILLIAM	L008	Systems	17
PC64	S31002	TULASI	L028	Database	23
PC64	S21010	PAVANI	L004	IT	17
PC64	S10034	BALU	L009	HR	16

SECOND NORMAL FORM (2NF)

Second normal form: A table is in the second normal form if it is in the first normal form and contains only columns that are dependent on the whole (primary) key.

STEPS:

Transform 1NF data into second normal form (2NF). Remove any -key attributes (partial Dependencies) that only depend on part of the table key to a new table. Ignore tables with a simple key or with no non-key attributes.

- The first table went straight to 2NF as it has a simple key (Project Code).
- Employee name, Department No and Department Name are dependent upon Employee No only. Therefore, they were moved to a new table with Employee No being the key.

- However, Hourly Rate is dependent upon both Project Code and Employee No as an employee may have a different hourly rate depending upon which project they are working on. Therefore it remained in the original table.

2NF Tables: Partial Key Dependencies Removed

Project Code	Project Title	Project Manager	Project Budget
PC10	MIS	Nisarga	24500
PC45	LIS	Eenchara	17400
PC64	HMS	Prakruthi	12250

Project Code	Emplyoee No	Hourly Rate
PC10	S10001	22
PC10	S10030	18
PC10	S21010	21
PC45	S10010	21
PC45	S10001	18
PC45	S31002	25
PC45	S13210	17
PC64	S31002	23
PC64	S21010	17
PC64	S10034	16

Emplyoee No	Employee Name	Department No	Department Name
S10001	ARUN	L004	IT
S10030	LOKESH	L023	Pionix
S21010	PAVANI	L004	IT
S10010	BABU	L004	IT
S31002	TULASI	L028	Database
S13210	WILLIAM	L008	Systems
S10034	BALU	L009	HR

THIRD NORMAL FORM (3NF)

Third normal form: A table is in the third normal form if it is in the second normal form and all the non-key columns are dependent only on the primary key. If the value of a non-key column is dependent on the value of another non-key column we have a situation known as transitive dependency. This can be resolved by removing the columns dependent on non-key items to another table.

STEPS:

Data in second normal form (2NF) into third normal form (3NF).
Remove to a new table any non-key attributes that are more dependent on other non-key attributes than the table key.

- The project team table went straight from 2NF to 3NF as it only has one non-key attribute.
- Department Name is more dependent upon Department No than Employee No and therefore was moved to a new table. Department No is the key in this new table and a foreign key in the Employee table.

3NF Tables: Non-Key Dependencies Removed

Project Code	Project Title	Project Manager	Project Budget
PC10	MIS	Nisarga	24500
PC45	LIS	Eenchara	17400
PC64	HMS	Prakruthi	12250

Project Code	Emplyoee No	Hourly Rate
PC10	S10001	22
PC10	S10030	18
PC10	S21010	21
PC45	S10010	21
PC45	S10001	18
PC45	S31002	25
PC45	S13210	17
PC64	S31002	23
PC64	S21010	17
PC64	S10034	16

Emplyoee No	Employee Name	Department No
S10001	ARUN	L004
S10030	LOKESH	L023
S21010	PAVANI	L004
S10010	BABU	L004
S31002	TULASI	L028
S13210	WILLIAM	L008
S10034	BALU	L009

Department No	Department Name
L004	IT
L023	Pionix
L028	Database
L008	Systems
L009	HR

Boyce-Codd Normal Form(BCNF)

A table is in Boyce-Codd normal form (BCNF) if and only if it is in 3NF and every determinant is a candidate key.

Anomalies can occur in relations in 3NF if there is a composite key in which part of that key has a determinant which is not itself a candidate key.

This can be expressed as R(A,B,C), C--->A where:

- The relation R contains attributes A, B and
C.
- A and B form a candidate key.
- C is the determinant for A (A is functionally dependent on C).
- C is not part of any key.

Anomalies can also occur where a relation contains several candidate keys where:

- The keys contain more than one attribute (they are composite keys).
- An attribute is common to more than one key.

Example to understand BCNF:-

Consider the following non-BCNF table:

Nearest Shops

Person	Shop Type	Nearest Shop
Ashwini	Optician	Eagle Eye
Ashwini	Hairdresser	Snippets
Chaya	Bookshop	GK Books
Namita	Bakery	Dlight
Namita	Hairdresser	Sweet Stall
Namita	Optician	Eagle Eye

The candidate key of the table are:

- {Person, Shop Type}
- {Person, Nearest Shop}

The table does not adhere to BCNF because of the dependency

Nearest Shop → Shop Type, in which the determining attribute (Nearest shop) is neither a candidate key nor a superset of a candidate key.

After Normalization.

Shop Near Person

Person	Shop
Ashwini	Eagle Eye
Ashwini	Snippets
Chaya	GK Books
Namita	Dlight
Namita	Sweet Stall
Namita	Eagle Eye

Shop

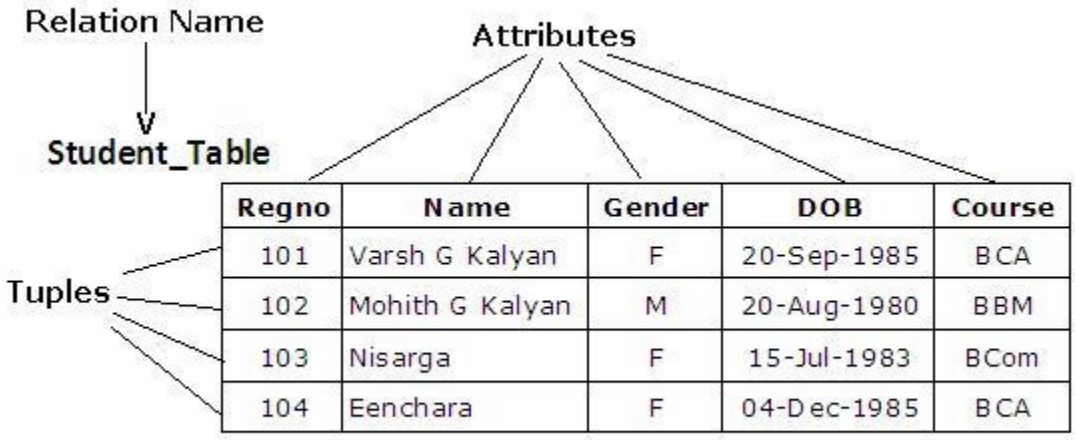
Shop	Shop Type
Eagle Eye	Optician
Snippets	Hairdresser
GK Books	Bookshop
Dlight	Bakery
Sweet Stall	Hairdresser
Eagle Eye	Optician

Candidate keys are {Person, Shop} and {Shop}, respectively.

Relational Data Model and Relational Algebra

Relational Model Concepts

The relational Model of Data is based on the concept of a Relation. A Relation is a mathematical concept based on the ideas of sets. The strength of the relational approach to data management comes from the formal foundation provided by the theory of relations. The model was first proposed by Dr. E.F. Codd of IBM in 1970 in the following paper: "A Relational Model for Large Shared Data Banks," Communications of the ACM, June 1970.



Regno	Name	Gender	DOB	Course
101	Varsh G Kalyan	F	20-Sep-1985	BCA
102	Mohith G Kalyan	M	20-Aug-1980	BBM
103	Nisarga	F	15-Jul-1983	BCom
104	Eenchara	F	04-Dec-1985	BCA

Fig: The attributes and tuples of a relation STUDENT_Table

Relation:

It is a table which has rows and columns in the data model, where rows represent records and columns represents the attributes.

Tuples:

A single row of a table, which contains a single record for that relation, is called a tuple.

Attributes:

Columns in a table are called attributes of the relation.

Cardinality of a relation: The number of tuples in a relation determines its cardinality. In this case, the relation has a cardinality of 4.

Degree of a relation: Each column in the tuple is called an attribute. The number of attributes in a relation determines its degree. The relation has a degree of 5.

Domain:

A domain definition specifies the kind of data represented by the attribute.

More- particularly, a domain is the set of all possible values that an attribute may validly contain. Domains are often confused with data types, but this is wrong. Data type is a physical concept while domain is a logical one. "Number" is a data type and "Age" is a domain. To give another example "StreetName" and "Surname" might both be represented as text fields, but they are obviously different kinds of text fields; they belong to different domains.

Properties of a Relation

A relation with **N** columns and **M** rows (tuples) is said to be of *degree* **N** and *cardinality* **M**. This is Student_Table which shows the relation of degree three and cardinality five.

Student_Table

Regno	Name	Course
101	Nisarga	BCA
102	Rama	BA
103	Eenchara	BBM
104	Prakruthi	B.Com
105	Eempana	BCA

Cardinality is 5

Degree is 3

The characteristic properties of a relation are as follows:

- ***All entries in a given column are of the same kind or type.***
- ***Attributes are unordered*** - The order of columns in a relation is immaterial. The display of a relation in tabular form is free to arrange columns in any order.

Student_Table

Regno	Name	Course
101	Nisarga	BCA
102	Rama	BA
103	Eenchara	BBM
104	Prakruthi	B.Com
105	Eempana	BCA

Student_Table

Regno	Name	Course
104	Prakruthi	B.Com
103	Eenchara	BBM
101	Nisarga	BCA
105	Eempana	BCA
102	Rama	BA

Column Ordering is not Important

- ***No duplicate tuples***. A relation cannot contain two or more tuples which have the same values for all the attributes. i.e., In any relation, every row is unique. ***There***
- ***is only one value for each attribute of a tuple***. The tuple should have only one value. The table shown below is not allowed in the relational model, despite the clear intended representation, ie. the Student has two values for Place, (eg. Nisarga has one in Pune, and one in Chennai). In such situations,the multiple values must be split into multiple tuples to be a valid relation.

Student_Table

Regno	Name	Place
101	Nisarga	Pune Chennai
102	Rama	Bangalore Mysore

- ***Tuples are unordered***. The order of rows in a relation is immaterial. One is free to display a relation in any convenient way.

Integrity Constraints over Relations

An integrity constraint (IC) is a condition that is specified on a database schema, and restricts the data that can be stored in an instance of the database. If a database instance satisfies all the integrity constraints specified on the database schema, it is a legal instance. A DBMS enforces integrity constraints, in that it permits only legal instances to be stored in the database.

Integrity constraints are specified and enforced at different times:

1. When the DBA or end user defines a database schema, he or she specifies the ICs that must hold on any instance of this database.
2. When a database application is run, the DBMS checks for violations and disallows changes to the data that violate the specified ICs.

Keys of a Relation

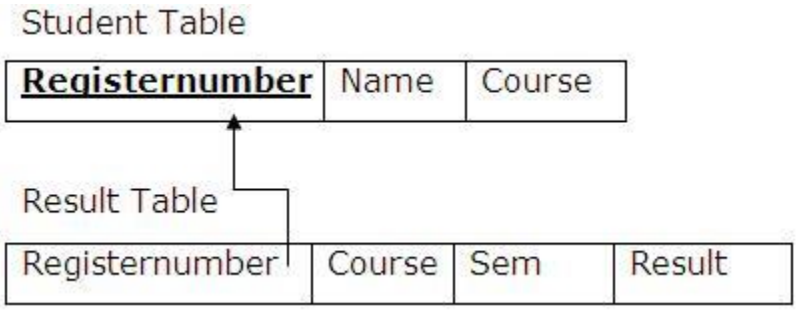
It is a set of one or more columns whose combined values are *unique* among all occurrences in a given table. A key is the relational means of specifying uniqueness. Some different types of keys are:

Primary key is an attribute or a set of attributes of a relation which posses the properties of uniqueness and irreducibility (No subset should be unique). For example: Register Number in Student table is primary key, Passenger Number in passenger table is primary key, Passport number in Booking table is a primary key and the combination of passenger number and Passport Number in Reservation table is a primary key ie composite primary key.

_____

Foreign key is the attributes of a table, which refers to the primary key of some another table. Foreign key permit only those values, which appears in the primary key of the table to which it refers or may be null (Unknown value).

For example: Register number of Result table refers to the Register number of Student table, which is the primary key of Student table, so we can say that Register number of Result table is the foreign key.



Relational Algebra

- Basic operations:
  - ✓ *Projection* (π) Selects a subset of columns from relation.
  - ✓ *Selection* (σ) Selects a subset of rows from relation.
  - ✓ *Cross-product* (×) Allows us to combine two relations.
  - ✓ *Set-difference* (-) Tuples in reln. 1, but not in reln. 2.
  - ✓ *Union* (U) Tuples in reln. 1 and in reln. 2.
  - ✓ *Rename*(ρ) Use new name for the Tables or fields.
- Additional operations:
  - ✓ Intersection (∩), *join*().

PROJECT (Π)

The PROJECT operation is used to select a subset of the attributes of a relation by specifying the names of the required attributes

Consider the Student_table:

Regno	Name	Gender	Course
101	Varsha G Kalyan	F	BCA
102	Mohith G Kalyan	M	BBM
103	Nisarga	F	Bcom
104	Rama	M	BCA

A) For example, to get a name from Student_Table.

πName(Student_Table)

Name
Varsha G Kalyan
Mohith G Kalyan
Nisarga
Rama

_____

B) For example, to get a regno and name from Student_Table.

$$\pi Regno,Name(Student\_Table)$$

Regno	Name
101	Varsha G Kalyan
102	Mohith G Kalyan
103	Nisarga
104	Rama

SELECT(σ)

The SELECT operation is used to choose a *subset* of the tuples from a relation that satisfies a **selection condition**. the SELECT operation can be consider to be a *filter* that keeps only those tuples that satisfy a qualifying condition.

A) For example, to list the regno > 102 from Student_Table.

$$\sigma_{Regno>102}(Student\_table)$$

Regno	Name	Gender	Course
103	Nisarga	F	Bcom
104	Rama	M	BCA

B) For example, to list all the Students belong to BCA course.

$$\sigma_{Course="BCA"}(Student\_table)$$

Regno	Name	Gender	Course
101	Varsha G Kalyan	F	BCA
104	Rama	M	BCA

Union, Intersection, Set-Difference

- All of these operations take two input relations, which must be union-compatible:
  - ✓ Same number of fields.
  - ✓ `Corresponding' fields have the same type.

Consider:

BORROWER

Cust-name	Loan_no
Ram	L-13
Shyam	L-30
Suleman	L-42

DEPOSITOR

Cust-name	Acc-No
Suleman	A-100
Radeshyam	A-300
Ram	A-401

UNION Operator

List of customers who are either borrower or depositor at bank

$\Pi_{\text{Cust-name}}$ (Borrower) U $\Pi_{\text{Cust-name}}$ (Depositor)

Cust-name
Ram
Shyam
Suleman
Radeshyam

INTERSECTION Operator

Customers who are both borrowers and depositors

$\Pi_{\text{Cust-name}}$ (Borrower) ∩ $\Pi_{\text{Cust-name}}$ (Depositor)

Cust-name
Ram
Suleman

Set Difference

Customers who are borrowers but not depositors

$\Pi_{\text{Cust-name}}$ (Borrower) - $\Pi_{\text{Cust-name}}$ (Depositor)

Cust-name
Suleman

Cartesian-Product or Cross-Product (S1 × R1)

- Each row of S1 is paired with each row of R1.
- *Result schema* has one field per field of S1 and R1, with field names `inherited' if possible.
- Consider the borrower and loan tables as follows:

BORROWER

Cust-name	Loan_no
Ram	L-13
Shyam	L-30
Suleman	L-42

Loan

Loan_no	Loan_Amt
L-13	10000
L-30	20000
L-42	40000

BorrowerCust-name	Borrower Loan_no	Loan Loan_No	Loan Loan_Amt
Ram	L-13	L-13	10000
Ram	L-13	L-30	20000
Ram	L-13	L-42	40000
Shyam	L-30	L-13	10000
Shyam	L-30	L-30	20000
Shyam	L-30	L-42	40000
Suleman	L-42	L-13	10000
Suleman	L-42	L-30	20000
Suleman	L-42	L-42	40000

JOIN

Join is combination of Cartesian product followed by selection process. Join operation pairs two tuples from different relations if and only if the given join condition is satisfied.

Following section describe briefly about join types:

Natural Join (⋈)

Natural Join can only be performed if there is at least one common attribute exists between relation. Those attributes must have same name and domain.

Natural join acts on those matching attributes where the values of attributes in both relation is same.

Courses		
CID	**Course**	**Dept**
CS01	Database	CS
ME01	Mechanics	ME
EE01	Electronics	EE

HoD	
Dept	**Head**
CS	Alex
ME	Maya
EE	Mira

Courses ⋈ HoD			
Dept	**CID**	**Course**	**Head**
CS	CS01	Database	Alex
ME	ME01	Mechanics	Maya
EE	EE01	Electronics	Mira

Theta (θ) join

Theta joins combines tuples from different relations provided they satisfy the theta condition.

Notation: $R1 \bowtie_\theta R2$

R1 and R2 are relations with their attributes (A1, A2, .., An) and (B1, B2,.. ,Bn) such that no attribute matches that is R1 ∩ R2 = Φ Here θ is condition in form of set of conditions C.

Theta join can use all kinds of comparison operators(=,<,>,≤,≥,≠).

Student		
SID	**Name**	**Std**
101	Alex	10
102	Maria	11

Subjects	
Class	**Subject**
10	Math
10	English
11	Music
11	Sports

Student_Detail = STUDENT ⋈Student.Std = Subject.Class SUBJECT

Student_detail				
SID	**Name**	**Std**	**Class**	**Subject**
101	Alex	10	10	Math
101	Alex	10	10	English
102	Maria	11	11	Music
102	Maria	11	11	Sports

Equi-Join

When Theta join uses only **equality** comparison operator it is said to be Equi-Join. The above example corresponds to equi-join.