UNIT 3 (Relational Model)

1) Write about CODD rules.

A) In 1985, Dr Edgar Frank Codd, A computer scientist working for IBM proposed the relational model for database management which forms the theoretical basis for relational databases. He defined thirteen rules, numbered from 0 to 12. Accordingly, if a database has to be called as true relational database management system, then it has to follow all these rules.

Rule zero

This rule states that for a system to qualify as an RDBMS, it must be able to manage database entirely through the relational capabilities.

Rule 1: Information rule

All the information including metadata (data about data) has to be represented as stored data in cells of tables.

Rule 2: Guaranteed Access Rule

Each unique piece of data(atomic value) should be accessible by Table Name + Primary Key(Row) + Attribute(column).

Rule 3: Systematic treatment of NULL

If any of the cell value is unknown, or not applicable or missing, it cannot be represent as zero or empty. It will be always represented as NULL. This NULL should be acting irrespective of the data type used for the cell.

Rule 4: Active Online Catalog Based On the Relational Model

Database dictionary (catalog) is the structure description of the complete Database and it must be stored online. We should be able to access these metadata by using same query language that we use to access the database.

Rule 5: Powerful and Well-Structured Language

One well structured language must be there to provide all manners of access to the data stored in the database. Example: SQL, etc. If the database allows access to the data without the use of this language, then that is a violation.

Rule 6: View Updation Rule

This rule states that views are also be able to get updated as we do with its table.

Rule 7: Relational Level Operation (High-Level Insert, Update And Delete Rule)

The system must support Insert, Update and Delete operations at each level of relations.

Rule 8: Physical Data Independence Rule

The physical storage of data should not matter to the system. If say, some file supporting table is renamed or moved from one disk to another, it should not effect the application.

Rule 9: Logical Data Independence

Here if there are any changes to the logical view, then it should not be reflected in the user view.

Rule 10: Integrity Independence Rule

The database should be able to enforce its own integrity rather than using other programs. Key and Check constraints, trigger etc, should be stored in Data Dictionary. This also makes RDBMS independent of front-end.

Rule 11: Distribution Independence
A database should work properly regardless of its distribution across a network. Even if a database is geographically distributed, with data stored in pieces, the end user should get an impression that it is stored at the same place. This lays the 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 integrity rules to change the data. This can be achieved by some sort of looking or encryption.

3) What is the difference between a database and a table?
Database : The database is a structure that contains one or more tables and metadata. A database is a place where we store required data. A database has tables of data, views, indexes and programs. A database can hold thousands of tables. The database is managed using tools called Database Management Systems(like Oracle, Informix, Sybase, DB2, SQL Server, MYSQL etc.,), you can create, view, modify, and delete databases. Different types of Databases are -Relational -Object Oriented -Object Relational.
Table : A table, a logical structure that represents an entity set, is only one of the components of a database. A table is perceived as two-dimentional structure composed of rows and columns. It exists inside a database. Relational database stores data in tables(called relations). These tables are related to each other. It is an object inside a database. What are the characteristics of Relational Table.

4) What is Entity Integrity and Referential Integrity.
A) **Entity integrity** is important, because it means that a proper search for an ***existing*** tuple (row) will always be successful. And the failure to find a match on a row search will always mean that the row for which the search is conducted does not exist in that table. **Referential integrity** is important, because its existence ensures that it will be impossible to assign a non-existing foreign key value to a table.

5) Write a short note on Keys
A) In relational model keys are very important because they are used to ensure that each row in a table is uniquely identifiable. They are also used to establish relationships among tables and to ensure the integrity of the data. Therefore, a proper understanding of the concept and use of keys in the relational model is very important. A key consists of one or more attributes that determine (find out) other attributes. Different type of keys are Super Key, Candidate Key, Primary Key, Secondary Key and Foreign Key.

7) Define the terms Super Key, Candidate Key, Primary Key, Composite-Primary-Key, Alternate Key, Secondary Key, Non-key attributes and Foreign Key.

A) **Super Key :** It is an attribute (or combination of attributes) that uniquely identifies each row in a table. In short, the super key functionally determines all of a row's attributes.
Ex: Consider STUDENT table consists of attributes namely STU_NUM,STU_LNAME, STU_FNAME, STU_INT, STU_PHONE, STU_DOB, STU_CLASS, STU_EMAIL. In this table Super Key could be any of the following.
STU_NUM (or)
STU_NUM,STU_LNAME (or)
STU_NUM,STU_LNAME,STU_FNAME (or)
STU_NUM,STU_LNAME,STU_FNAME,STU_INT.
In fact, STU_NUM, with or without additional attributes, can be a Super Key even when the additional attributes are surplus.

Candidate Key : A candidate key can be described as a super key without unnecessary attributes, that is, a minimal (irreducible) super key.
Ex : In the above example STU_NUM is considered as a candidate key since it is a minimal super key.

Primary Key : A candidate key selected to uniquely identify all other attribute values in any given row and cannot contain null entries. (or) A primary key is a set of one or more attributes that can uniquely identify tuples within the relation.
Ex : In the above example STU_NUM is considered as primary key.

Composite-Primary-Key : A group of attributes(more than one) is considered as a Composite-Primary-Key, if it serves the purpose of a primary key, that is, group of attributes provides a unique value which is used to uniquely identify rows in a table.
Ex : Let us consider a table CLASS with attributes CRS_CODE, CLASS_SECTION, CLASS_TIME, PROF_CODE). Here no attribute alone can be taken as primary key so the combination of CRS_CODE and CLASS_SECTION is treated as composite-primary-key.

Candidate Key : All attribute combinations inside a relation that can serve as primary key are candidate keys as they are candidates for the primary key positions.
Ex : Let us consider a table SUPPLIER with attributes SUP_NUM, SUP_NAME, STATUS, CITY. Here SUP_NUM and SUP_NAME are eligible for candidate keys, because SUP_NUM and SUP_NAME alone can serve the purpose of a primary key.

Note : In case of two or more candidate keys, the database designer decides one of them as the primary key for the relation according to requirement.

Alternate Key : A candidate key that is not the primary key is called as an Alternate key.
Ex : In Supplier example SUP_NAME is treated as alternate key if SUP_NUM is considered as primary key and vice-versa.

Secondary Key : An attribute (or combination of attributes) used strictly for data retrieval purposes.
Ex : Let us consider a table CUSTOMER with attributes CUST_NUM, CUST_LNAME, CUST_FNAME, CUST_PHONE, CUST_ZIP. Consider CUST_NUM is primary key. While searching the data, Instead of using CUST_NUM as key we may use CUST_LNAME and CUST_PHONE combination as key which is considered as secondary key.

Non-Key Attributes : The non-primary-key attributes of a table are referred to as non-key attributes.
Ex : In the above CUSTOMER table except CUST_NUM all attributes are treated as non-key attributes.

Foreign Key : An attribute (or combination of attributes) in one table whose values must either math the primary key in another table or be null. It is used to represent the relationship between two tables.
(or) A non-key attribute, whose values are derived from the primary key of some other table (or be null), is known as Foreign-Key in its current table.
Ex : Consider two tables COURSE with attributes CRS_CODE, DEPT_CODE, CRS_DESC and CLASS with attributes CRS_CODE, CLASS_SEC, CLASS_TIME. Here CRS_CODE is primary key in COURSE table and foreign-key in CLASS table. Here foreign-key contains either matching values or nulls.

8) What is Datadictionary and System Catalog?
A) **Data Dictionary** : The Data Dictionary contains meta data – data about data. The Data Dictionary provides a detailed description of all tables found within the user (or designer) created database. It contains at least all of attribute's names and characteristics for each table in the system. It is sometimes described as "the database designer's database" because it records the design decisions about tables and their structures.
System Catalog : The System catalog is actually a system created database whose tables store the user (or designer) created database characteristics and contents. Like the Data Dictionary, the System Catalog contains metadata. It is described as the detailed system data dictionary that describes all objects within the database, including data about the table names, the table's creator and creation date, the number of columns in each table, the data type corresponding to each column, index filenames, index creators, authorized users, and access privileges.

9) Write about Relational Algebra.
A) Relational algebra is a widely used **procedural query language**. It collects instances of relations as input and gives occurrences of relations as output. It uses various operation to perform this action.

10) Write about various types of operations of Relational Algebra.
A) Every database management system must define a query language to allow users to access the data stored in the database. Relational Algebra is a procedural query language used to query the database tables to access data in different ways. In relational algebra, input is a relation(table from which data has to be accessed) and output is also a relation(a temporary table holding the data asked for by the user).
Relational Algebra works on the whole table at once, so we do not have to use loops etc to iterate over all the rows(tuples) of data one by one. All we have to do is specify the table name from which we need the data, and in a single line of command, relational algebra will traverse the entire given table to fetch data.
The primary operations that we can perform using relational algebra are:
Select
Project
Union
Set Different
Cartesian product
Rename
Relational algebra operations are performed recursively on a relation. The output of these operations is a new relation, which might be formed from one or more input relations.
Relational Algebra divided in various groups
Unary Relational Operations

SELECT (symbol: **σ**) - sigma

PROJECT (symbol: ∏) - pi

RENAME (symbol: **ρ**) - rho

Relational Algebra Operations From Set Theory
UNION (∪)
INTERSECTION (∩),
DIFFERENCE (-)

CARTESIAN PRODUCT (X)

Binary Relational Operations
JOIN(⋈)
DIVISION

SELECT (σ) - It selects ROWS based on given condition(predicate)
The SELECT operation is used for selecting a subset of the tuples according to a given selection condition. Sigma(σ)Symbol denotes it. Select operation selects tuples that satisfy a given predicate.
$^{\sigma}$ p(r)
σ is the predicate
r stands for relation which is the name of the table
p is prepositional logic

Example:1
σ topic = "Database" (Tutorials)
Output - Selects tuples from Tutorials where topic = 'Database'.

Example 2
σ topic = "Database" and author = "guru"(Tutorials)
Output - Selects tuples from Tutorials where the topic is 'Database' and 'author' is guru.

Example 3
σ marks>85 (Student)
Output - Selects tuples from Student where marks is greater than 85.

Projection(∏)
The projection eliminates all attributes of the input relation but those mentioned in the projection list. The projection method defines a relation that contains a vertical subset of Relation. (pi) The symbol used to choose attributes from a relation. This operation helps you to keep specific columns from a relation and discards the other columns.
Ex: Table : Customer

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

If we want to get CustomerName and Status of Customer table we use projection as follows.
∏CustomerName, Status (Customers)
Output:

CustomerName	Status
Google	Active
Amazon	Active
Apple	Inactive
Alibaba	Active

Union operation (∪)
UNION is symbolized by ? symbol. It includes all tuples that are in tables A or in B. It also eliminates duplicate tuples. So, set A UNION set B would be expressed as:
The result <-- A **U** B

For a union operation to be valid, the following conditions must hold -
R and S must be the same number of attributes.
Attribute domains need to be compatible.
Duplicate tuples should be automatically removed.

Example : Consider the following tables A,B for UNION, DIFFERENCE & INTERSECTION.

Table A

col1	col2
1	1
1	2

Table B

col1	col2
1	1
1	3

A ∪B **gives**

Table : A ∪ B

col1	col2
1	1
1	2
1	3

Set Difference (-)

- Symbol denotes it. The result of A - B, is a relation which includes all tuples that are in A but not in B.

The attribute name of A has to match with the attribute name in B.
The two-operand relations A and B should be either compatible or Union compatible.
It should be defined relation consisting of the tuples that are in relation A,but not in B.
Example A-B
Table A - B

col1	col2
1	2

Intersection
An intersection is defined by the symbol n
A ∩ B
Defines a relation consisting of a set of all tuple that are in both A and B. However, A and B must be union-compatible.
Example: A ∩ B
Table A ∩ B

col1	col2
1	1

Cartesian product(X)
This type of operation is helpful to merge columns from two relations. Generally, a Cartesian product is never a meaningful operation when it performs alone. However, it becomes meaningful when it is followed by other operations.

Example – Cartesian product
σ col2 = '1' (A X B)

Output – The above example shows all rows from relation A and B whose col2 has value 1

σ col2 = '1' (A X B)

col1	col2
1	1
1	1

Types of JOIN:
Various forms of join operation are:
Inner Joins:
- Theta join
- EQUI join
- Natural join

Outer join:
- Left Outer Join
- Right Outer Join
- Full Outer Join

Inner Join:
In an inner join, only those tuples that satisfy the matching criteria are included, while the rest are excluded. Let's study various types of Inner Joins:
Theta Join($_\theta$):
The general case of JOIN operation is called a Theta join. It is denoted by symbol ?
Example : A $\bowtie_\theta$ B
Theta join can use any conditions in the selection criteria.
For example:
A $\bowtie_\theta$ A.col2 > B.col2 (B)
A $\bowtie_\theta$ A.col2 > B.col2 (B)

col1	col2
1	2

EQUI join(=):
When a theta join uses only equivalence condition, it becomes a equi join.
For example:
A ⋈ A.col2 = B.col2 (B)
A ⋈ A.col2 = B.col2 (B)

col1	col2
1	1

EQUI join is the most difficult operations to implement efficiently in an RDBMS and one reason why RDBMS have essential performance problems.

NATURAL JOIN (⋈)

Natural join can only be performed if there is a common attribute (column) between the relations. The name and type of the attribute must be same.

Example

Consider the following two tables

C

Num	Square
2	4
3	9

D

Num	Cube
2	8
3	18

C ⋈ D

Num	Square	Cube
2	4	4
3	9	9

OUTER JOIN

In an outer join, along with tuples that satisfy the matching criteria, we also include some or all tuples that do not match the criteria.

Left Outer Join(A ⟕ B)

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

Consider the following 2 Tables

A

Num	Square
2	4
3	9
4	16

B

Num	Cube
2	8
3	18
5	75

A ⟕ B

Num	Square	Cube
2	4	4
3	9	9
4	16	-

Right Outer Join: (A ⟖ B)

In the right outer join, operation allows keeping all tuple in the right relation.

However, if there is no matching tuple is found in the left relation, then the attributes of the left relation in the join result are filled with null values.

$A \bowtie B$

Num	Cube	Square
2	8	4
3	18	9
5	75	-

Full Outer Join: (A ⟗ B)

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

$A \bowtie B$

Num	*Cube*	*Square*
2	4	8
3	9	18
4	16	-
5	-	75 .

11) Write about Relational Calculus.

A) Relational Calculus in **non-procedural query language** and has no description about how the query will work or the data will be fetched. It only focuses on what to do, and not on how to do it. It is contrary to Relational Algebra which is a procedural query language to fetch data and which also explains how it is do,

Relational Calculus exists in two forms:
Tuple Relational Calculus (TRC)
Domain Relational Calculus (DRC)

Tuple Relational Calculus (TRC)

In tuple relational calculus, we work on filtering tuples based on the given condition.
Syntax:
{ T | Condition }
In this form of relational calculus, we define a tuple variable, specify the table(relation) name in which the tuple is to be searched for, along with a condition.
We can also specify column name using a . dot operator, with the tuple variable to only get a certain attribute(column) in result.
A tuple variable is nothing but a name, can be anything, generally we use a single alphabet for this, so let's say T is a tuple variable.
To specify the name of the relation(table) in which we want to look for data, we do the following:
Relation(T), where T is our tuple variable.

Ex: if our table is Student, we would put it as Student(T)
Then comes the condition part, to specify a condition applicable for a particular attribute(column), we can use the . dot variable with the tuple variable to specify it, like in table Student, if we want to get data for students with age greater than 17, then, we can write it as,
T.age > 17, where T is our tuple variable.
Putting it all together, if we want to use Tuple Relational Calculus to fetch names of students, from table Student, with age greater than 17, then, for T being our tuple variable,
Ex1:
Fetch names of students, from table Student, with age greater than 17
T.name | Student(T) AND T.age > 17

Ex2:
Returns tuples with 'name' from Author who has written article on 'database'.
T.name | Author(T) AND T.article = 'database' }

Domain Relational Calculus (DRC)
In domain relational calculus we use list of attribute to be selected from the relation based on the condition. It is same as TRC, but differs by selecting the attributes rather than selecting whole tuples. It is denoted as below:
Syntax:
{ c1, c2, c3, ..., cn | F(c1, c2, c3, ... ,cn)}
Domain relational calculus uses the same operators as tuple calculus. It uses logical connectives ∧ (and), ∨ (or) and ¬ (not).
Where c1, c2, c3, … cn are attributes of the relation and F is the condition and F defines the formula including the condition for fetching the data.

Ex1:
{< name, age > | ∈ Student ∧ age > 17}
The above query will return the names and ages of the students in the table Student whose age is greater than 17.

Ex2:
{< article, page > |∈ Oracle ∧ topic=’CDB’}
The above query will return the article and page columns of Oracle table where topic is ‘CDB’.

Ex3:
{< article, page, subject > | ∈ books ∧ subject = 'database'}
The above query will return the article, page, and subject from the relation books, where the subject is a database.

12) What is QBE?
A) Query by example is a query language used in relational databases that allows users to search for information in tables and fields by providing a simple user interface where the user will be able to input an example of the data that he or she wants to access. The principle of QBE is that it is merely an abstraction between the user and the real query that the database system will receive. In the background, the user's query is transformed into a database manipulation language form such as SQL, and it is this SQL statement that will be executed in the background.