**Overview of Database Management**

1. What is database?

A database is a logical collection of data with some inherent meaning, and which is designed, built and populated with data for a specific purpose.

2.What is DBMS?

It is a collection of programs that enables user to create and maintain a database. In other words it is general-purpose software that provides the users with the processes of defining, constructing and manipulating the database for various applications.

3. What is a Database System?

The database and DBMS software together is called as Database system.

**Database Language(SQL) categorization :**

Applications interact with the relational database through a special language called SQL(STRUCTURED QUEY LANGUAGE) which includes :

DDL(Create, Alter, Drop)

DML(Insert, Update, Delete)

DQL(Select)

TCL(Commit, Rollback, Savepoint)

DCL(Grant, Revoke)

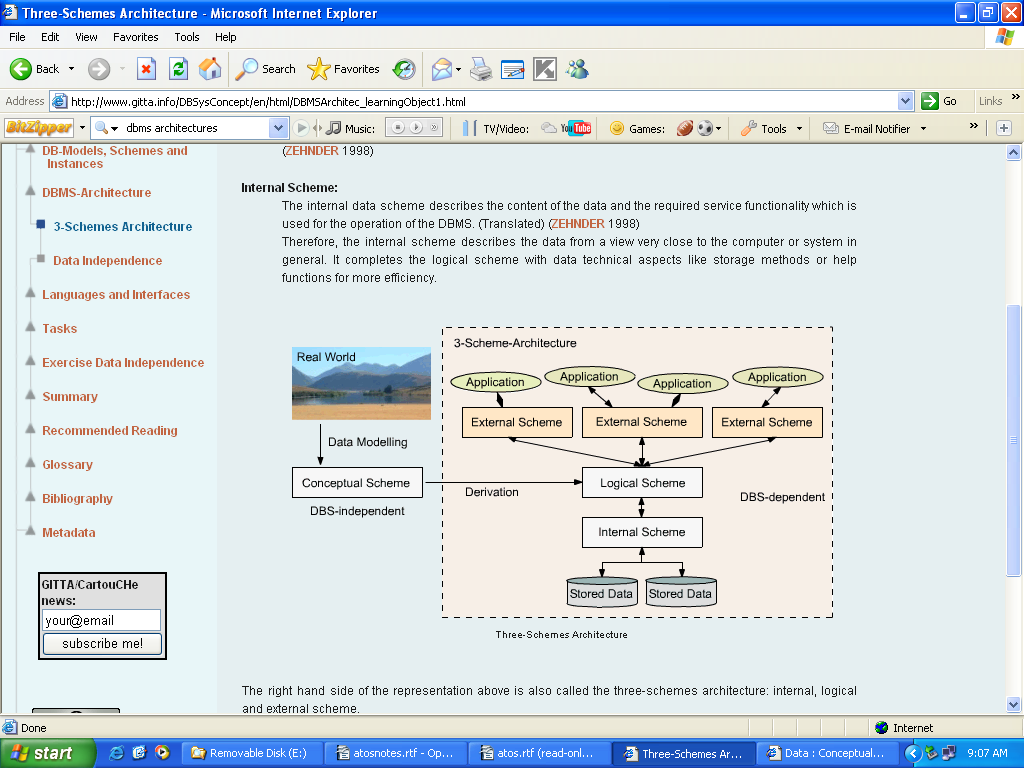
**RDBMS architecture**

3-Tier Architecture:

The **internal schema(Physical schema)** describes the physical grouping of the data and the use of the storage space. This schema level decides the way how the data is actually physically stored on some storage medium

The **conceptual(logical) schema** describes the basic construction of the data structures or database objects. This schema level describes what data are stored in the database in terms of objects, and what relationships exist among those objects

The **external schema(application)** of a specific application, generally, only highlights that part of the logical scheme which is relevant for its application.



**Mapping or Data Independence**

Mapping means that the implementation of the External, Logical(Conceptual) and the Internal(Physical) schemas/schemes should be done in such a way, using the tools/objects of the DBMS, that changes in the Internal/Physical structure should not at all or by very minor efforts affect the Logical(Conceptual) layer and changes in the Logical(Conceptual) layer should not at all or by very minor efforts affect External layer. Achieving this independence between the Internal/Physical and Conceptual/logical layer is called Physical Independence, and independence between the Conceptual/logical layer and External/Application layer is called Logical Independence.

**Two types of Data Independence:**

**Physical Data Independence**: Achieving independence between the Internal/Physical and Conceptual/logical layer is called Physical Independence

Example, if a user’s data was earlier on a single physical file and now is spread across two or more such physical files, this physical or internal representation should not make any difference to how the user accesses his data via the SQL/PLSQL

**Logical Data Independence**: Achieving independence between the Conceptual/logical layer and External/Application layer is called Logical Independence.

Example

TAB1 FRAG1 FRAG2

**A B C D A B A C D**

1 A C E 1 A 1 C E

4 A C F 4 A 4 C F

6 B D G 6 B 6 D G

2 B D H 2 B 2 D H

It should be possible to split a table vertically into more than one fragment as long as such splitting preserves all the original data.

FRAG1 FRAG2 TAB1

**A B A C D A B C D**

1 A 1 C E 1 A C E

4 A 4 C D 4 A C D

6 B 6 D G 6 B D G

2 B 2 D H 2 B D H

It should be possible to combine several base tables into one by way of preserving all the original data.

Both, can be achieved to a great extent using **Views.**

**Database Schema and Database Instance**

**Database Schema** is the overall design/description of the database, and is **not frequently subject to changes in a running system**. That is, it describes the database in terms of the various users, tables, and other database objects at a given point of time.

**Database Instance** is the state of the data in the database at a given moment of time, and is always subject to a lot of changes in an online/running system. That is, it describes the data in the tables of the database at a given point of time.

**Analytic and Operational Databases**

Analytic databases used for OLAP(On Line Analytical Processing) are primarily **static, read-only databases**(periodically updated from the OLTP database) which store archived, historical data used for analysis. For example, a bank might store customer transaction records over the last ten years in an analytic database and use that database to analyze banking strategies or trends to give better service to it’s customers.

Operational databases used for OLTP(On Line Transaction Processing), are used to manage more **dynamic** bits of data or current or live data(frequently updated by the transaction in an up and running system). Operational databases allow you to modify data (add, change or delete data). These types of databases are usually used to track real-time information. For example, a Bank might have an operational database used to track day-to-day/daily transaction of their customers, new customer entries, existing customer-detail changes, etc

**Relational Model**

* A Database consists of multiple Relations or Tables
* Information about an enterprise is broken up into parts, with each relation or table storing one part of the information
* Each part is termed as Table and rows of a table are termed as Tuples and columns as Attributes

Properties of Relational Tables:

* Values in columns should be Atomic
* Each Row should be Unique
* Column Values are of the Same Kind
* The Sequence of Columns is Insignificant
* The Sequence of Rows is Insignificant
* Each Column has a Unique Name

A relational database is a **collection of 2-dimensional tables which consist of rows and columns. Tables are known as "relations" or “entity sets”; columns are known as "attributes" or “fields”; and rows (or records) are known as "tuples" or “entities”.**

Tables / Relations are a **logical structure therefore they are an abstract concept and they do not represent how the data is stored on the physical computer system. Each column / attribute in a relation represents an attribute of an entity. A single row / tuple contains all the information (in the form of attributes) about a single entity.**

**The order of records and columns is irrelevant. Relations/Tables and columns should always be uniquely named and therefore uniquely identifiable.** No duplicate rows should occur in a single relation.

**Each cell of a relation contains a single value or element which is atomic. This means that arrays or lists, for example, would not be stored under a single attribute. (Though in OORDBMSs, this is implemented, example, collections in Oracle)**. Multi-valued attributes are possible though but this involves a technique of referring to another relation which holds these multiple values.

**Object-Oriented Relational Model**

* Built on top of relational data models by including object orientation and constructs to deal with added data types
* Allow attributes of tuples to have complex types, including non-atomic values such as nested relations
* Provide a convenient migration path for users of RDBMS who wish to use OO features
* It encapsulates methods with data structures like columns(object columns) and tables(object tables)
* Compatibility with existing relational languages such as SQL

**KEYS concept in RDBMS/Relational Model**

A **Key** is a single attribute or combination of two or more attributes of an entity set that is used to identify one or more instances of the set.

PRIMARY KEY:-A primary key is a field that uniquely identifies each record in a table. As it uniquely identify each entity, it cannot contain null value and duplicate value.

**Super Key**

Any combination of attributes that can uniquely identify an entity is called a Super Key. The default Super Key is combination of all the columns of a table. A table has many Super Keys

**Candidate Key**

A minimal Super Key is called candidate Key. By minimal we mean if you drop any component from the key then it no more remains unique. A table may have multiple candidate keys

Example 1:

======

Imagine a table with the fields <Name>, <Age>, <SSN> and <Phone Extension>.

This table has many possible superkeys.

Three of these are

<SSN>

<SSN,Phone Extension>

<SSN, Name>.

Of those listed, only <SSN> is a candidate key, as the others contain information not necessary to uniquely identify records.

Example 2:

======

Given table:

EMPLOYEES{employee\_id, firstname, surname, sal}

Possible superkeys are:

{employee\_id}

{employee\_id, firstname}

...

(employee\_id, firstname, surname, sal}

Only the **minimal superkey** - {employee\_id} - will be considered as a **candidate key**.

ALTERNATE KEY:-A candidate key that is not the primary key is called an Alternate key.

COMPOSITE KEY:- Creating a primary key jointly from more than one attribute is known as composite key.

FOREIGN KEY:- A foreign key is one or more columns whose value must exist in the primary key of another table or same table

**Domain/Data/Value Constraints**

A domain defines the possible values of an attribute. Domain/Value **Constraints**(Check, Unique, Not Null) govern these values. In a database system, the domain integrity is defined by:

The datatype and the length

The NULL value acceptance

The allowable values, through techniques like check/unique/not null constraints

The default value

For example, if you define the attribute of Age, of an Employee entity, is an integer, the value of every instance of that attribute must always be numeric. If you also define that this attribute must always be positive, then a negative value is forbidden. The value of this attribute being compulsory indicates that the attribute cannot be NULL. All of these characteristics form the domain/Value constraints of this attribute.

**Integrity Constraints**

Entity Integrity Constraint: States that “Primary key cannot have NULL or Duplicate value”

Referential Integrity Constraint: States that “Foreign Key can be either a NULL value or should be a value from the Primary Key to which it points.

**DATABASE STORAGE HIERARCHY**

**1) DATABASE**

A collection of data files integrated and organised into a simple comprehensive file system, which is arranged to minimize duplication of data and to provide convenient access to information within that system to satisfy a wide variety of user needs.

**2) DATA FILES**

Datafile is the file which actually physically keeps the data in secondary storage device. It represents the physical representation of the database.

**3) DATA OBJECTS(Tables)**

Objects such as tables, procedures, functions, etc, especially TABLES logically keep the data stored together. These objects may me spread across one or more physical data files. Tables or Relations are Entity Sets.

**4) RECORDS**

A collection of related items of data that are treated as a unit. It is the information about an entity.

eg:- An employee record would be collection of all fields of one employee.  
Record is sometimes referred as tuple.

**5) FIELD**

Individual element of data is called Field.  
eg:- Bank cheque consist of following field cheque no, date, payee, numeric amt, signature, bank,etc.

Field is sometimes referred as Data item Or column or attribute.

**Basic RDBMS TERMS**

**1) RELATION**

A relationship is an association among several entities  
eg:- A cusst\_Acct relationship associates a customer with each account that she or he has.

Also, a table or an entity set is sometimes called a relation.

In both cases, a relation is a rectangular output consisting of row(tuples or records) and columns(attributes of fields)

**2) REDUNDANCY**

If same piece of information is stored in database for several number of times, the database is said to be redundant. We should check our database should not be redundant as it wastes disk space, reduced efficiency of database, require more processing time, and their are chances of inconsistency due to it in our database.  
eg:-If we have to tables emp\_details (contains details of employee) and Payroll(contains Payment details to employee), then if we include details of employee in payroll table, than it is said to be redundancy as same piece of information is repeated.

**3) INCONSISTENCY**

Inconsistency is various copies of the same data which is not consistent. Inconsistency occurs due to redundancy, so redundancy should be reduced.

eg:- If we have details of employee stored in emp\_details and payroll table than while updating information we should check that both tables are updated or not, if we update the address of one employee in emp\_details and same details is not updated in payroll table, than database is said to be in inconsistent state.

**4) PROPAGATING UPDATES**

Propagating updates ensures that changes made to records/data of one relation or tables, are automatically made to other tables or relations. This process is known as Propagating updates. Where the term "Updates" is used to cover all the operations of insertion, deletion and modification.   
We can avoid inconsistency by using propagating update technique.

**6)USERS**

There are broadly three different types of database system users.

**End Users**

They are users who interact with the system by invoking one of the application programs. Thus, they are persons who use the information generated by a computer based system. Retrieval is the most common function for this class of user. Generally application user, data entry operators and report generators are come under this category.

**Application Programmers**

They are who prepare or code the application. Application programs operates on the data in all the usual ways: retrieving information, inserting new information, deleting or changing existing information.

**Database Administrator**

These are the highly skilled users responsible for the overall performance and maintenance of the database. They design the database schema, create the database, create users, grant and revoke rights from users and manage the database. Their responsibility includes recovery of database in case of failure and tuning the database for optimum efficiency.

**Roles**

A Role is a collection of rights which can be allocated to one or more users of the Database. It helps in minimizing the administrative tasks/maintenance of the DBAs.

# Entity Relationship Model and Diagram

# ERD or Entity Relationship Diagram is a pictorial representation of the various entities in a database and the relationship between them. ERD represents the logical(conceptual) database as viewed by end user

ERDs depict the ER model’s three main components:

Entities

Attributes

Relationships

Entities and the Relationships between them are shown using following conventions(Chen Model) :

-An **entity** is shown as a **rectangle**.

-A **diamond** represents the **relationships** among a number of entities, which are connected to the diamond by **lines**.

-The **attributes**, shown as **ovals**, are connected to the entities or relationships by lines.

-Diamonds, rectangles and ovals are **labeled**. The type of relationship existing between the entities is represented by giving the **cardinality** of the relationship on the line joining the relationship to the entity.

Attr1

Attr1

Attr1

N

ENTITY2

ENTITY1

M

RELATIONSHIP

Attr2

Attr2

Attr2

**Entity**

An entity is an object that is of interest to an organization. Entities or Objects of similar types are characterised by some **common set of properties or attributes**. Such similar entities or objects form an **entity set or entity type**. Two objects or entities are distinguishable and this fact is represented in the entity set by giving them a **Unique identifier(s)**.

# Attributes

# They are characteristics of entities. *Domain* is a set of possible values for a particular attribute. Key or Unique Identifier or Primary key is underlined or boldfaced in the ER diagram. Attributes can be of following types :

### Simple

# Cannot be subdivided such as Age, sex, etc

# Composite

# Can be subdivided such as Address consists of street, city, state, zip

# Single-valued

# Has only a single value such as Social security number

# Multi-valued

# Can have many values. For example a person may have several college degrees

# Derived

# Can be calculated from other information like age can be derived from D.O.B., total salary can be derived from Basic, DA, TA, etc

# Relationships(Cardinality ratio)

# They are associations between entities which are established by Business Rules. The Connected entities are termed as participants. The Connectivity describes relationship classification(1:1, 1:M, M:N). Cardinality(Mapping Cardinalities expresses the number of entities of one table or entity-set to which another entity of another table or entity-set can be associated via a relationship set) is the number of entity occurrences associated with one occurrence of related entity.

**Degree of Relationship**

A relationships degree indicates the number of associated entities.

A relationship is called a **Binary Relationship**, if the number of entity sets involved is two.

M

N

STUDENT

COURSE

ENROLLMENT

However, the entity sets involved in a relationship may not always be distinct entity sets. You may call them **Unary Relationship**.

EMPLOYEE

Manager Subordinates

Manages

1

N

PERSON

1

1

SPOUSE

SPOUSE

Marries

PERSON

PARENTS

CHILDREN

M

N

OFFSPRINGS

A relationship that involves N entities is called **N-ary Relationship**.

Following is an example of Ternary Relationship.

STUDYING represents the relationship between a STUDENT studying a COURSE through a particular STUDYCENTRE.

STUDYCENTRE

STUDYING

COURSE

STUDENT

n



Identification of a relationship is done by using the Primary Keys of the entities involved in it. Thus, if Relationship R involves Entity Sets E1, E2, E3,….Ek having Primary Keys p1, p2, p3,… pk, the unique identifier of the relationship R is the composite attribute(p1, p2, p3,…pk). An instance of the relationship R is represented by concatenating it’s attribute(r1, r2, …. rm) with the primary keys of the instances of the entities involved in the relationship.

**12 Codds commandments/rules**

**Codd's twelve rules** are a set of thirteen rules ([numbered zero to twelve](https://en.wikipedia.org/wiki/Zero-based_numbering)) proposed by [Edgar F. Codd](https://en.wikipedia.org/wiki/Edgar_F._Codd), a pioneer of the [relational model](https://en.wikipedia.org/wiki/Relational_model) for [databases](https://en.wikipedia.org/wiki/Database), designed to define what is required from a [database management system](https://en.wikipedia.org/wiki/Database_management_system) in order for it to be considered *relational*, i.e., a relational database management system ([RDBMS](https://en.wikipedia.org/wiki/RDBMS))

**Rule 0:** The *Foundation rule*:

A relational database management system must manage its stored data using only its relational capabilities. The system must qualify as [*relational*](https://en.wikipedia.org/wiki/Relational_model), as a [*database*](https://en.wikipedia.org/wiki/Database), and as a [*management system*](https://en.wikipedia.org/wiki/Management_system). For a system to qualify as a [relational database management system](https://en.wikipedia.org/wiki/Relational_database_management_system) (RDBMS), that system must use its *relational* facilities (exclusively) to *manage* the [*database*](https://en.wikipedia.org/wiki/Database).

**Rule 1:** The *information rule*:

All information in a relational database (including table and column names) is represented in only one way, namely as a value in a table.

**Rule 2:** The *guaranteed access rule*:

All data must be accessible. This rule is essentially a restatement of the fundamental requirement for [primary keys](https://en.wikipedia.org/wiki/Unique_key). It says that every individual scalar value in the database must be logically addressable by specifying the name of the containing [table](https://en.wikipedia.org/wiki/Table_(database)), the name of the containing column and the primary key value of the containing [row](https://en.wikipedia.org/wiki/Row_(database)).

**Rule 3:** *Systematic treatment of null values*:

The DBMS must allow each field to remain null (or empty). Specifically, it must support a representation of "missing information and inapplicable information" that is [systematic](https://en.wiktionary.org/wiki/systematic), distinct from all regular values (for example, "distinct from zero or any other number", in the case of numeric values), and independent of [data type](https://en.wikipedia.org/wiki/Data_type). It is also implied that such representations must be handled/manipulated by the DBMS in a systematic way.

**Rule 4:** *Active* [*online*](https://en.wikipedia.org/wiki/Online)[*catalog*](https://en.wikipedia.org/wiki/Database_catalog) *based on the relational model* **The database description rule**

The system must support an online, inline, relational [catalog](https://en.wikipedia.org/wiki/Database_catalog) that is accessible to authorized users by means of their regular [query language](https://en.wikipedia.org/wiki/Query_language). That is, users must be able to access the database's structure (catalog) using the same query language that they use to access the database's data.

**Rule 5:** The *comprehensive data sublanguage rule*:

The system must support at least one relational language that, namely, structured query language

1. Has a [linear syntax](https://en.wikipedia.org/wiki/Linear_syntax)
2. Can be used both interactively and within application programs,
3. Supports data definition operations (including view definitions), data manipulation operations (update as well as retrieval), security and integrity constraints, and [transaction](https://en.wikipedia.org/wiki/Database_transaction) management operations (begin, commit, and rollback).

**Rule 6:** The [*view*](https://en.wikipedia.org/wiki/View_(database)) *updating rule*:

All views that are theoretically updatable must be updatable by the system.

**Rule 7:** *High-level insert, update, and delete*/ **The Insert and Update rule**

The system must support set-at-a-time *insert*, *update*, and *delete* operators. This means that data can be retrieved from a relational database in sets constructed of data from multiple rows and/or multiple tables. This rule states that insert, update, and delete operations should be supported for any retrievable set rather than just for a single row in a single table.

**Rule 8:** *Physical data independence*:

Changes to the physical level (how the data is stored, whether in the form of arrays or linked lists etc. in one or more physical files) must not require a change to the way the data is accessed through commands or through any application.

**Rule 9:** *Logical data independence*:

Changes to the logical level (tables, columns, rows, and so on) must not require a change to an application based on the logical structure. Logical data independence is more difficult to achieve than physical data independence. Views can be used to a good extent to achieve this

**Rule 10:** *Integrity independence*:

[Integrity constraints](https://en.wikipedia.org/wiki/Integrity_constraints) must be specified separately from application programs and stored in the [catalog](https://en.wikipedia.org/wiki/Database_catalog). It must be possible to change such constraints as and when appropriate without unnecessarily affecting existing applications.

**Rule 11:** *Distribution independence*:

The end-user must not be able to see that the data is distributed over various locations. Users should always get the impression that the data is located at one site only. This rule has been regarded as the foundation of distributed database systems

**Rule 12:** The *nonsubversion rule*:

If a system has an interface that provides access to low-level records, then the interface must not be able to subvert the system and bypass security and integrity constraints.

**What is a Transaction?**

A collection of operations that perform a single logical unit of work or functionality in a database application

Following are the Properties of transaction, it is also call ACID properties of transaction:

### Atomicity

If one part of the transaction fails, the entire transaction fails, and the database state is left unchanged. An atomic system must guarantee atomicity in each and every situation, including power failures, errors, and crashes.

### Consistency

The [consistency](https://en.wikipedia.org/wiki/Consistency_(database_systems)) property ensures that any transaction will bring the database from one valid state to another. Any data written to the database must be valid according to all defined rules, including [constraints](https://en.wikipedia.org/wiki/Integrity_constraints), [triggers](https://en.wikipedia.org/wiki/Database_trigger), and any combination thereof. This does not guarantee correctness of the transaction, that is the responsibility of application-level code.

### Isolation

The [isolation](https://en.wikipedia.org/wiki/Isolation_(database_systems)) property ensures that the concurrent execution of transactions results in a system state that would be obtained if transactions were executed serially, i.e., one after the other. Providing isolation is the main goal of [concurrency control](https://en.wikipedia.org/wiki/Concurrency_control). Depending on concurrency control method, the effects of an incomplete transaction might not even be visible to another transaction.

### Durability

[Durability](https://en.wikipedia.org/wiki/Durability_(computer_science)) means that once a transaction has been committed, it will remain so, even in the event of power loss, [crashes](https://en.wikipedia.org/wiki/Crash_(computing)), or errors. In a relational database, for instance, once a group of SQL statements execute, the results need to be stored permanently (even if the database crashes immediately thereafter).

**NORMALISATION**

Normalization is a process of organizing your data in the form of tables in such a way that :

1) data is stored in the database with minimum usage of hard-disk capacity

2) There is no unnecessary duplication of data

3) There are no insertion, updation and deletion anomalies

4) Data abides by all the business rules implemented through constraints

**Functional Dependency**

In a table or relation, if a given value of attribute X can determine the value of attribute Y, then the attribute X is called the determinant and the attribute Y is said to be functionally dependent on attribute X. This functional dependency is denoted as X → Y, which means attribute X decides attribute Y, that is, for a given value of X, there is one value of Y

Consider the EMP table following Relation, for a given value of EMPNO, there is one value of ENAME, SAL, COMM, DEPTNO, HIREDATE, MGR, JOB. All these functional dependencies are indicated as :

EMPNO-->ENAME, EMPNO-->SAL, EMPNO-->COMM, EMPNO-->DEPTNO, EMPNO-->HIREDATE, EMPNO-->MGR, EMPNO-->JOB

Mostly, in a table, the determinants are candidate keys or key attribute. However, in the given table, the Marks attribute is not a key attribute, but marks determines grade, considering that 90 and above is Grade A, 70 to 89 is Grade B, 50 to 69 is Grade C, and 49 or less Grade D. Hence it can be concluded that key attributes are determinants but not all the determinants are key attributes.

**Rollno StudentName Course Marks Grade**

1 Tom Maths 91 A

2 Peter Maths 76 B

3 John Maths 58 C

4 Michel Maths 70 B

5 Hary Maths 49 D

1 Tom Science 77 B

2 Peter Science 96 A

3 John Science 38 D

4 Michel Science 75 B

5 Hary Science 59 C

Also, Rollno and Course together (called composite key attribute) defines EXACTLY ONE value of marks. This can be symbolically represented as **Rollno,Course → Marks**. Thus, a determinant can be composite in nature

Consider the following data which shows preferences of faculty members for teaching courses. Here, we allows cross-departmental teaching, that is, a faculty member of Computer Science department may have preference for teaching a course of the Mathematics department. The following data is in UNNORMALISED form, since each row may contain multiple set of values for certain columns. Such multiple values for a given column in a single row is called NONATOMIC values. For example, the row corresponding to the preferences of the faculty in the Computer Science Department has two values for the column Professor. For a particular professor say professor Smith, the row indicating his preferences contains three values for the column Course.

Such UNNORMALISED relation contains NONATOMIC values.

|  |  |  |  |
| --- | --- | --- | --- |
| PROF\_DEPT | PROF | COURSE | COURSE\_DEPT |
| Computer Science | Smith | 353 | Computer Science |
| 379 | Computer Science |
| 221 | Decision Science |
| Clark | 353 | Computer Science |
| 351 | Computer Science |
| 379 | Computer Science |
| 456 | Mathematics |
| Chemistry | Turner | 353 | Computer Science |
| 456 | Mathematics |
| 272 | Chemistry |
| Mathematics | James | 353 | Computer Science |
| 379 | Computer Science |
| 221 | Decision Science |
| 456 | Mathematics |
| 469 | Mathematics |

Hence, the above unnormalised data is converted to the following format, so that it does not contain non-atomic values.

|  |  |  |  |
| --- | --- | --- | --- |
| PROF\_DEPT | PROF | COURSE | COURSE\_DEPT |
| Computer Science | Smith | 353 | Computer Science |
| Computer Science | Smith | 379 | Computer Science |
| Computer Science | Smith | 221 | Decision Science |
| Computer Science | Clark | 353 | Computer Science |
| Computer Science | Clark | 351 | Computer Science |
| Computer Science | Clark | 379 | Computer Science |
| Computer Science | Clark | 456 | Mathematics |
| Chemistry | Turner | 353 | Computer Science |
| Chemistry | Turner | 456 | Mathematics |
| Chemistry | Turner | 272 | Chemistry |
| Mathematics | James | 353 | Computer Science |
| Mathematics | James | 379 | Computer Science |
| Mathematics | James | 221 | Decision Science |
| Mathematics | James | 456 | Mathematics |
| Mathematics | James | 469 | Mathematics |

FNF

A Relation is said to be in the FIRST NORMAL FORM or 1NF, if the values in the domain of each attribute of the relation are ATOMIC. In other words, only one value is associated with each attribute of a row, and the value is not a list of values(as in the above example) or a composite value(as shown in the example below).

In this example, the data is not un-normalised, but the column Order\_details contains composite values(value of order number and value of order date)

|  |  |  |  |
| --- | --- | --- | --- |
| Order\_details | Item\_code | Quantity | Price\_per\_unit |
| 1456 of 26/02/1989 | 3687 | 52 | 50.40 |
| 1456 of 26/02/1989 | 4627 | 38 | 60.20 |
| 1456 of 26/02/1989 | 3214 | 20 | 17.50 |
| 1886 of 04/03/1992 | 4629 | 45 | 20.25 |
| 1886 of 04/03/1992 | 4627 | 30 | 60.20 |
| 1788 of 04/04/1992 | 4627 | 40 | 60.20 |

FNF

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Order\_No | Order\_date | Item\_code | Quantity | Price\_per\_unit |
| 1456 | 26/02/1989 | 3687 | 52 | 50.40 |
| 1456 | 26/02/1989 | 4627 | 38 | 60.20 |
| 1456 | 26/02/1989 | 3214 | 20 | 17.50 |
| 1886 | 04/03/1992 | 4629 | 45 | 20.25 |
| 1886 | 04/03/1992 | 4627 | 30 | 60.20 |
| 1788 | 04/04/1992 | 4627 | 40 | 60.20 |

The representation of data for the courses that a faculty member would prefer to teach has the following drawbacks : The fact that a given professor belongs to a particular department is repeated a number of times. The fact that a particular course is offered by a particular department is also repeated a number of times. These repetitions can lead to inconsistencies. For example, if a Professor’s department is changed, unless all the rows where that professor appears are changed, we would have inconsistencies in the database. Also, if the relation between courses and the departments to which the courses belong is kept in this table only, then a new course cannot be entered(without NULL values, since may be all the four columns of the above table form a composite primary key) unless some professor prefers to teach it. Also, deletion of the only professor who prefers to teach a given course will cause loss of information of the fact that such a course exists, and also loss of the information about the department to which this course belongs.

SNF

A Relation is said to be in the SECOND NORMAL FORM or 2NF, if it is in the FNF and if all the non-key(non-prime) attributes are fully(wholly) functionally dependent on the relation’s key attribute(s). It ensures that there are no Partial dependencies

Consider the following relation :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Order\_No | Order\_date | Item\_code | Quantity | Price\_per\_unit |
| 1456 | 26/02/1989 | 3687 | 52 | 50.40 |
| 1456 | 26/02/1989 | 4627 | 38 | 60.20 |
| 1456 | 26/02/1989 | 3214 | 20 | 17.50 |
| 1886 | 04/03/1992 | 4629 | 45 | 20.25 |
| 1886 | 04/03/1992 | 4627 | 30 | 60.20 |
| 1788 | 04/04/1992 | 4627 | 40 | 60.20 |

Here, the key of the above relation is Order\_no+Item\_code(Composite Key). The following Functional Dependencies exist :{Order\_No 🡪 Order\_date, Item\_code 🡪 Price\_per\_unit, Order\_No+Item\_code 🡪Quantity}.

Here, the attribute Order\_date which is not a part of the key of the relation is called non-key attribute, and is not functionally dependent on the whole key(Order\_No+Item\_code), but is dependent on a part of the whole key, that is on Order\_No. Also, same is the case with the attribute Price\_per\_unit, which is dependent on a part of the whole key, that is on Item\_code, Only, the attribute Quantity is fully functionally dependent on the whole key(Order\_No+Item\_code).

This relation has several problems :

Redundancy

In the above table, which is in 1NF, the Order\_date and Price\_per\_unit are repeated several times. The aim of a database system is to reduce redundancy, meaning that the information should be stored only once. Storing information several times leads to wastage of storage space and also increase in the total size of the data stored. Such redundant data can lead to inconsistency, as explained in the next point

Update Anomalies

Multiple copies of the same data may lead to update anomalies or inconsistencies when an update is made and only some of the multiple copies of the redundant data are updated. Thus, a change in the Price\_per\_unit must be made, for consistency, in all tuples pertaining to the item. If any one of the tuples is not changed to reflect the new Price\_per\_unit, there will be inconsistency in the data.

Insertion Anomalies

If this is the only table that shows the relation between an order and the items ordered in that item, then the fact that a given item exists cannot be entered in the database unless an order is placed for that item.

Deletion Anomalies

If an order is cancelled or deleted, and it is the only order that includes a given item, then the fact that item exists is also lost

Therefore, it is split into 3 tables as follows to give 2NF :

|  |  |
| --- | --- |
| Order\_No | Order\_date |
| 1456 | 26/02/1989 |
| 1886 | 04/03/1992 |
| 1788 | 04/04/1992 |

Now order\_date(non key attribute) is functionally dependent on order\_no(key attribute)

|  |  |
| --- | --- |
| Item\_code | Price\_unit |
| 3687 | 50.40 |
| 4627 | 60.20 |
| 3214 | 17.50 |
| 4629 | 20.25 |

Price\_unit(non key attribute) is functionally dependent on Item\_code(key attribute)

|  |  |  |
| --- | --- | --- |
| Order\_no | Item\_code | Qty |
| 1456 | 3687 | 52 |
| 1456 | 4627 | 38 |
| 1456 | 3214 | 20 |
| 1886 | 4629 | 45 |
| 1886 | 4627 | 30 |
| 1788 | 4627 | 40 |

Qty(non key attribute) is functionally dependent on Order\_no+Item\_code(key attribute)

TNF

A Relation is said to be in the THIRD NORMAL FORM or 3NF, if it is **does not allow Transitive Dependencies,** that is, No non-key attributes are functionally dependent on any other non-key attribute

Consider the following in 2NF:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Roll\_No | Name | Department | Year | Hostel\_name |
| 1784 | Ram | Physics | 1 | Ganga |
| 1648 | Sham | Chemistry | 1 | Ganga |
| 1768 | Bharat | Maths | 2 | Kaveri |
| 1848 | Arjun | Botany | 2 | Kaveri |
| 1682 | Nakul | Geology | 3 | Krishna |
| 1485 | Sahadeo | Zoology | 4 | Godavari |

Assuming that the 1st Year, 2nd Year, 3rd Year and 4th Year students stay in Hostels Ganga, Kaveri, Krishna and Godavari respectively, then in such a case, the non-key attribute Hostel\_name is dependent on the non-key attribute Year. Therefore, it is split into 2 tables to avoid unnecessary duplication of data as follows to give 3NF :

Stud\_mast

|  |  |  |  |
| --- | --- | --- | --- |
| Roll\_No | Name | Department | Year |
| 1784 | Ram | Physics | 1 |
| 1648 | Sham | Chemistry | 1 |
| 1768 | Bharat | Maths | 2 |
| 1848 | Arjun | Botany | 2 |
| 1682 | Nakul | Geology | 3 |
| 1485 | Sahadeo | Zoology | 4 |

Hostel\_Mast

|  |  |
| --- | --- |
| Year | Hostel\_name |
| 1 | Ganga |
| 2 | Kaveri |
| 3 | Krishna |
| 4 | Godavari |

**3.4.4. Boyce Codd Normal Form (BCNF)**

A relation is said to be in Boyce Codd Normal Form (BCNF) if and only if all the determinants are candidate keys. BCNF relation is a strong 3NF, but not every 3NF relation is BCNF.

Consider the following Result table structure.

**St**

Student# EmailID Course# Marks

**101** [Davis@myuni.edu](mailto:Davis@myuni.edu) **M4 82**

**102** [Daniel@myuni.edu](mailto:Daniel@myuni.edu) **M4 62**

**101** [Davis@myuni.edu](mailto:Davis@myuni.edu) **H6 79**

**103** [Sandra@myuni.edu](mailto:Sandra@myuni.edu) **C3 65**

**104** [Evelyn@myuni.edu](mailto:Evelyn@myuni.edu) **B3 77**

**102** [Daniel@myuni.edu](mailto:Daniel@myuni.edu) **P3 68**

**105** [Susan@myuni.edu](mailto:Susan@myuni.edu) **P3 89**

**103** [Sandra@myuni.edu](mailto:Sandra@myuni.edu) **B4 54**

**105** [Susan@myuni.edu](mailto:Susan@myuni.edu) **H6 87**

**104** [Evelyn@myuni.edu](mailto:Evelyn@myuni.edu) **M4 65**

In the RESULT table, we have two candidate keys namely **Student#Course#** and **EmaiIId**.**Course#.** Course# is overlapping among those candidate keys. Hence these candidate keys are called as **“overlapping candidate keys”**

The non-key attribute, Marks is non-transitively and fully functionally dependant on key attributes. Hence this is in 3NF. But this is not in BCNF because there are four determinants in this relation namely:

Student# (Student# decides EmailiD)

EMailID (EmailID decides Student#)

Student# Course# (decides Marks)

 EMailID Course# (decides Marks)

All above determinants are not candidate keys. EMailID decides Student# but EMailID on its own is not a candidate key. Similarly Student# decides EMailID of a student but Student# alone is not a candidate key. Only combination of (Student#, Course#) and (EMailID, Course#) are candidate keys. However, Student# decides EmailID and EMailID decides Student#, hence the duplication of data.

To make this table BCNF, we need to split this table into the following structure:

**STUDENT TABLE**

**Student# EmailID**

**Student # EmailID**

**101** Davis@myuni.edu

**102** Daniel@myuni.edu

**103** Sandra@myuni.edu

**104** Evelyn@myuni.edu

**105** Susan@myuni.edu

**Student**

**Student # Course# Marks**

**101 M4 82**

**102 M4 62**

**101 H6 79**

**103 C3 65**

**104 B3 77**

**102 P3 68**

**105 P3 89**

**103 B4 54**

**105 H6 87**

**104 M4 65**

Now both the tables are not only in 3NF, but also in BCNF because all the determinants are candidate keys. In the first table, Student# decides EMailID and EMailID decides Student# and both are candidate keys.

In second table, Student# Course# is only determinant and candidate key. Hence it qualifies BCNF definition that every determinant must be a candidate key.

If the table has only one non-composite/composite candidate key and if it is in 3NF, then the table will also be in BCNF.

Basically 2NF and 3NF takes away the redundancy, anomalies which exist among the key and non-key attributes respectively. On other hand BCNF takes away the redundancy, anomalies which exist amongst the key attributes

Let us assume the following reality

* For each subject, each student is taught by only one Instructor
* Each Instructor teaches only one course
* Each course is taught by several Instructors

Course Instructor Student Marks

C1 James Rahul 78

C1 James Rohan 87

C1 James Ramesh 56

C1 Gomes Suresh 45

C1 Gomes Sohan 34

C2 Yen Rahul 98

C2 Pen Suresh 56

For a given value of **Course**, **Student**  there is one value of Instructor, and one value of Marks

For a given value of **Student, Instructor**  there is one value of Course, and one value of Marks

For a given value of **Instructor**  there is one value of Course

The column STUDENT is overlapping in the two possible candidate keys. The two possible candidate keys are (**Course**, **Student)**  and (**Student, Instructor).** Hence these candidate keys are called as **“overlapping candidate keys”**

The combination of(Instructor, Course) cannot determine a student, as multiple students can have that combination, and also marks.

The table is in 2nd Normal form as the non-key Marks is fully functionally dependent on the candidate keys (**Course**, **Student)**  and (**Student, Instructor).** Also, there is no non-key dependent on another non-key. Despite of this, there is redundancy, as for a given value of Instructor there is one value of Course. This is because neither in 2nd or 3rd normal forms, we look at the interdependency between candidate keys, especially, composite candidate keys, when they are overlapping candidate keys. Hence, we need BCNF

There are 3 determinants in this relation namely:

Instructor (Instructor decides course)

Course, Student (decides rest of the attributes in RESULT table)

Student, Instructor (decides rest of the attributes in RESULT table)

All above determinants are not candidate keys. Instructor decides Course but Instructor on its own is not a candidate key. Only combination of (**Course**, **Student)**  and (**Student, Instructor).**  are candidate keys.

Hence, if we know the instructor name, we automatically know the course name as an Instructor teaches only one course.

Instructor Student Marks

James Rahul 78

James Rohan 87

James Ramesh 56

Gomes Suresh 45

Gomes Sohan 34

Yen Rahul 98

Pen Suresh 56

Course Instructor

C1 James

C1 Gomes

C2 Yen

C2 Pen

* BCNF refers to decompositions involving Relations with more than one candidate key, where the candidate keys are composite and overlapping
* That is, a relation is in BCNF if and only if every determinant is a candidate key

**Fourth Normal Form :**

A Boyce Codd normal form relation is in fourth normal form if

1. there is no multivalued dependency(**a --->> b)** in the relation or
2. there are multivalued dependencies but the attributes, which are multivalue dependent on a specific attribute, are dependent between themselves.

Assume the following relation

**R(a,b,c) having columns a, b and c and together the combination (a,b,c) is the key of the table**

Recall that a relation is in BCNF if all its determinant are candidate keys, in other words each determinant can be used as a primary key.

Because relation **R** has only one determinant **(a, b, c)**, which is the composite primary key, therefore R is in BCNF.

Now R may or may not be in fourth normal form.

If R contains **no multi valued dependency** then R will be in Fourth normal form.

Assume R has the following two-multi value dependencies:

**a --->> b and a --->> c**

In this case R will be in the fourth normal form if **b** and **c dependent on each other.**

However if b and **c are independent of each other** then **R is not** in **fourth** normal form and the relation has to be projected to form two non-loss projections. These non-loss projections will be in fourth normal form.

Assume the following relation with multi-value dependency:

Employee (Eid, Language, Skill) where (Eid, Language, Skill) is the composite primary key.

Multivalued Dependency is symbolically indicated as follows : X --->> Y, which means for a given value X, there are multiple values of Y.

In our case,:

Eid --->> Languages Eid --->> Skills

Languages and Skills are dependent.

This says an employee speaks several languages and has several skills. However for each skill, a specific language is used when that skill is practiced

|  |  |  |
| --- | --- | --- |
| Eid | Language | Skill |
| 100 | English | Teaching |
| 100 | Kurdish | Singing |
| 100 | French | Cooking |
| 200 | English | Cooking |
| 200 | Arabic | Singing |

**Thus employee 100 when he/she teaches speaks English but when he cooks, he cooks French. This relation is in fourth normal form and does not suffer from any anomalies.**

Now, assume :

Eid --->> Languages Eid --->> Skills

Languages and Skills are Independent.

This relation is not in fourth normal form and suffers from all three types of anomalies.

|  |  |  |
| --- | --- | --- |
| Eid | Language | Skill |
| 100 | English | Teaching |
| 100 | Kurdish | Singing |
| 100 | English | Singing |
| 100 | Kurdish | Teaching |
| 200 | Arabic | Singing |

**Insertion anomaly** :

To insert row (200 English) we have to insert the rows (200 English Singing) as we cannot keep skill empty.

Hence

|  |  |  |
| --- | --- | --- |
| 200 | Arabic | Singing |
| 200 | English | Singing |

To insert (200, Story-telling), we have to insert (200, English, Story-telling) and (200, Arabic, Story-telling) as Language cannot be kept empty.

Hence

|  |  |  |
| --- | --- | --- |
| 200 | Arabic | Singing |
| 200 | English | Singing |
| 200 | Arabic | Story-telling |
| 200 | English | Story-telling |

**Deletion anomaly: If employee 100 discontinue Singing skill we have to delete two rows:**

**(100 Kurdish Singing), and (100 English Singing) otherwise the database will be inconsistent.**

**Updation Anomaly** : Update anomaly: If employee 200 changes his skill from singing to dancing we have to make changes in more than one place.

Hence, the relation/table is projected to the following two non-loss projections which are in forth normal form

|  |  |
| --- | --- |
| Eid | Language |
| 100 | English |
| 100 | Kurdish |
| 200 | Arabic |
| 200 | English |

AND

|  |  |
| --- | --- |
| Eid | Skill |
| 100 | Teaching |
| 100 | Singing |
| 200 | Singing |
| 200 | Story-telling |
|  |  |

**Fifth Normal Form**

A relation R is in Fifth Normal Form (5NF) if and only if the following conditions are satisfied simultaneously:

1.         R is already in 4NF.

2.         It cannot be further non-loss decomposed.

A table is in fifth normal form (5NF) or Project-Join Normal Form (PJNF) if it is in 4NF and it cannot have a lossless decomposition into any number of smaller tables.

**Properties of 5NF:-**

* Anomalies can occur in relations in 4NF if the primary key has three or more fields.
* 5NF is based on the concept of join dependence - if a relation cannot be decomposed any further then it is in 5NF.
* Pair wise cyclical dependency means that:
  + For any one you must know the other two (cyclical).

***Example to understand 5NF***

Take the following table structure as an example of a buying table. This is used to track buyers, what they buy, and from whom they buy. Take the following sample data:

Buyer Vendor Item

B1 V1 I1

B1 V1 I2

B1 V2 I3

B2 V1 I2

B2 V2 I4

**Problem:-** The problem with the above table structure is that if Vendor V1 starts to sell I3 then how will you record this fact? The problem is there are pair wise cyclical dependencies in the primary key. That is, in order to determine/record the item you must know the buyer and vendor, and to determine the vendor you must know the buyer and the item, and finally to know the buyer you must know the vendor and the item.

**Solution:-** The solution is to break this one table into three tables; Buyer-Vendor, Buyer-Item, and Vendor-Item. So following tables are in the 5NF.

* **Buyer-Item**

Buyer Item

B1 I1

B1 I2

B1 I3

B2 I2

B2 I4

* **Vendor-Item**

Vendor Item

V1 I1

V1 I2

V2 I3

V2 I4

* **Buyer-Vendor**

Buyer Vendor

B1 V1

B1 V2

B2 V1

B2 V2

**Denormalization**

A normalized design will often store different but related pieces of information in separate logical tables (called relations). If these relations are stored physically on separate disk files, writing a query that draws information from several relations can be slow. If many relations are joined, it may be prohibitively slow. There are two strategies for dealing with this.

1) The preferred method is to keep the logical design normalized, but allow the DBMS to store additional redundant information on disk to optimize query response. This method is often implemented in as indexed views or materialized views.

2) The other approach is to denormalize the logical data design. With proper care, this can achieve a similar improvement in query response, but at a cost―it is now the database designer's responsibility to ensure that the denormalized database does not become inconsistent. This is done by creating rules in the database called triggers, that specify how the redundant copies of information must be kept synchronized.