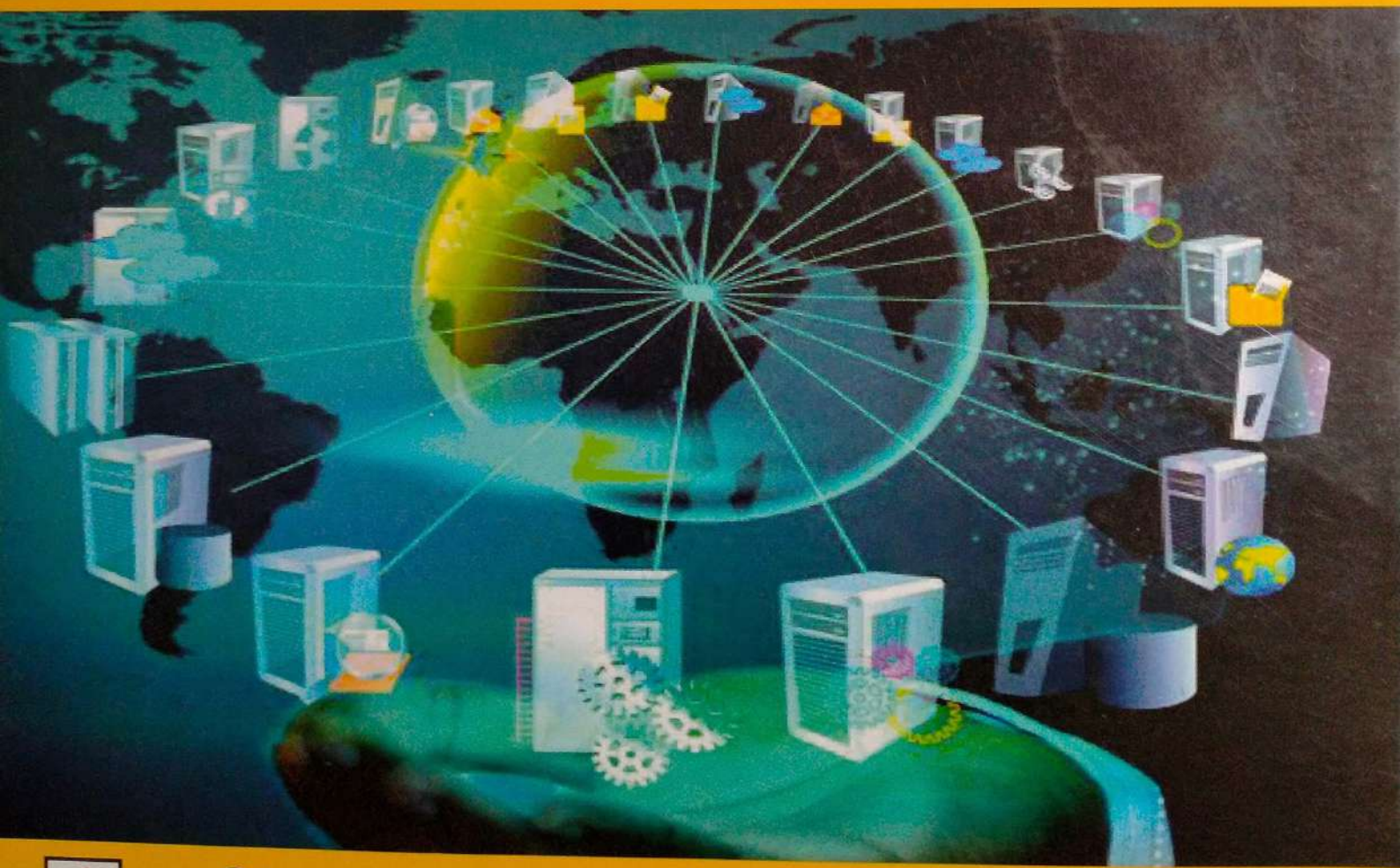
प्रियम तायल • पवन कुमार गुप्ता • प्रद्युम्न कुमार

डाटाबेस प्रबन्धन प्रणाली

DATABASE MANAGEMENT SYSTEM



एशियन पब्लिशर्स, मुज़फ़्फ़रनगर®

प्राविधिक शिक्षा परिषद्, उत्तर प्रदेश द्वारा स्वीकृत, नवीन संशोधित पाठ्यक्रमानुसार

डाटाबेस प्रबन्धन प्रणाली

(DATABASE MANAGEMENT SYSTEM)

चतुर्थ सत्र (द्वितीय वर्ष) कम्प्यूटर साइंस इंजीनियरिंग एवं इन्फॉर्मेशन टेक्नोलॉजी डिप्लोमा के छात्रों के लिए

Vertexal.

लेखक:

प्रियम तायल

असिस्टेंट प्रोफेसर

गेटवे इन्स्टीट्यूट ऑफ इंजीनियरिंग एण्ड टेक्नोलॉजी

सोनीपत (हरियाणा)

पवन कुमार गुप्ता

विभागाध्यक्ष, सूचना प्रौद्योगिकी (आई०टी०)

चौ० मुख्तार सिंह राजकीय बालिका पॉलीटेक्निक

दौराला, मेरठ (उत्तर प्रदेश)

प्रद्युम्न कुमार

भूतपूर्व आई०टी० विश्लेषक, टी०सी०एस० लि०

प्रवक्ता, एम०एम०आई०टी

संत कबीर नगर (उत्तर प्रदेश)

2023-2024

प्रकाशक:

एशियन पब्लिशर्स, मुज़फ़्फ़रनगर®

[illegible], नई मण्डी, मुज़फ़्फ़रनगर–251 001 (उ०प्र०)

डाटाबेस प्रबन्धन प्रणाली

• प्रियम तायल • पवन कुमार गुप्ता • प्रद्युम्न कुमार

प्रकाशकः

एशियन पब्लिशर्स, मुज़फ़्फ़रनगर®

46/20, कम्बल वाला बाग, नई मण्डी,
मुज़फ़्फ़रनगर – 251 001 (उ०प्र०)
फोन: 0131-2660989
Visit us at : www.asianpublishers.co.in
email : spmittal@asianpublishers.co.in
sales@asianpublishers.co.in

प्रथम संस्करण : 2020-2021
पुनःमुद्रित : 2021, 2022
पुनःमुद्रित : 2023-2024

मूल्य : ₹ 200.00

ISBN: 978-93-5502-144-1

लेजर टाइपसैटिंगः
सारा एसाईनमैन्ट्स
शहादरा, दिल्ली
फोन: 011-22572589

मुद्रकः
विमल प्रिन्टर्स
मेरठ
फोन: 0-9412203584

प्रस्तावना

लेखकों का इस पुस्तक को लिखने का इरादा विशेषरूप से डिप्लोमा के विद्यार्थियों के लिए डाटाबेस प्रबन्धन प्रणाली की बुनयादी पाठ्यपुस्तक को सरल भाषा में बनाने का है। इस पुस्तक को लिखते समय बहुत ही सरल भाषा का का उपयोग किया गया है तथा टेक्निकल शब्दों को यथासंभव अंग्रेजी भाषा में भी देने का प्रयास किया है। इस पुस्तक की विषय वस्तु को उत्तर प्रदेश स्टेट बोर्ड ऑफ़ टेक्निकल एजुकेशन के द्वारा प्रतिपादित नवीनतम पाठ्यक्रम के अनुसार इस प्रकार प्रस्तुत किया गया है कि एक सामान्य विद्यार्थी भी बिना किसी तनाव के समझ कर याद कर सकता है। इस पुस्तक के प्रत्येक पाठ के अंत में अभ्यास प्रश्नों को सलंगित किया गया है जिसकी मदद से विद्यार्थी पाठ का संशोधन और अभ्यास कर सकते हैं।

लेखक अपने सभी सहयोगियों, शुभचिन्तकों, दोस्तों और साथी अध्यापकों का हार्दिक धन्यवाद प्रस्तुत करते हैं जिन्होंने इस पुस्तक को लिखने के दौरान सतत मदद और सहयोग प्रदान किया। लेखक अपने प्रकाशक, मेसर्स एशियन पब्लिशर्स, मुज़्ज़फ़रनगर (उत्तर प्रदेश) का भी दिल की गहराई से धन्यवाद प्रस्तुत करते हैं जिन्होंने लेखकों के ऊपर विश्वास किया हैं।

अन्त में, लेखक उन विद्यार्थियों या अध्यापकों की सराहना करते हैं जो भविष्य में इस पुस्तक में सुधार के लिए अपने विचार, आलोचनाएं, टिप्पणी और समीक्षा भेजेंगे।

—लेखक

SYLLABUS

DATABASE MANAGEMENT SYSTEM

L T P
5 – 4

RATIONALE

The diploma holders in Computer Science and Engineering need to understand about Relational Data base to manage the data at backend for different applications. They should be able to develop basic table and write query to fetch the required data. Hence this subject.

LEARNING OUTCOMES

After undergoing the subject, students will be able to:

- understand the concept of Database system and Client Server Architecture
- understand and develop the concepts of Data Modeling, Security and Integrity.
- convert and compare the designs and differentiate between the keys
- understand and execute different SQL queries and PL/SQL programs
- convert database in the form of table
- normalize the database using normal forms.
- understand the concept of query processing and Transaction processing

SUGGESTED DISTRIBUTION OF MARKS

S. No.	Topics	Time Allotted (Periods)	Marks Allotted (%)
1.	Database System Concept & Data Modeling	10	15
2.	Data Model	10	15
3.	Relation Model	10	15
4.	Relational Database Design	11	17
5.	MYSQL/SQL	11	17
6.	PL-SQL	10	10
7.	NO-SQL	03	04
8.	Security	05	07
	Total	70	100

DETAILED CONTENTS

1. Database System Concept and Data Modeling (10 Periods)

Basic concepts, Advantages of a DBMS over file processing system, Data Abstraction, Database Languages, Data Independence, Components of a DBMS and overall structure of a DBMS, Three views of Data (External View, Conceptual View, Internal View), Three level architecture of DBMS, Data Independence, Client Server Architecture.

2. **Data Model** (10 Periods)

Define data model, Data Models: Network Model Hierarchical Model, E-R Model, Advantage & Disadvantages of each Data Model

ER Model: Entity sets and relationship sets—Attributes—Keys in entity and relationship sets: (a) Super Key (b) Candidate Key (c) Primary Key (e) Unique Key—Mapping constraints, Participation Constraint, E-R diagram, Notations. Strong Entity Set and Weak Entity Set.

3. **Relation Model** (10 Periods)

Advantages, Disadvantages, Codd's 12 rules, Definition of Relations, Schema, Sub schema. Relational Model Constraints (Domain, Tuple Uniqueness, Key Constraints, Integrity Constraints, Entity constraints).

Relations algebra (Basic operation: Union intersection difference and Cartesian product), Additional Relational Algebraic Operations (Projection, Selection rows, Division, rename and join), Converting ER Model to Relational Model.

4. **Relational Database Design** (11 Periods)

Purpose of Normalization, Data redundancy and updating anomalies, Functional Dependencies and Decomposition, Process of Normalization using 1NF, 2NF, 3NF, multivalued dependencies and BCNF, Forth Normal Form, Fifth Normal Form.

5. **MYSQL/SQL** (11 Periods)

Data definition language, Data manipulation language, SQL, Object naming conventions, Object naming guidelines, Data types, Tables (Creating, Inserting, Updating and deleting tables and using constraints), Views, Indexes.

SQL Command: DESCRIBE, SELECT, WHERE CLAUSE, DISTINCT CLAUSE, ORDER BY, HAVING, LOGICAL OPERATIONS, SQL OPERATORS, JOIN

Aggregate functions, String functions and date time functions, Null values.

6. **PL-SQL** (10 Periods)

User defined function, Control of flow statement of PL/SQL, Procedures/Stored procedures, transaction, triggers, cursors, granting and revoking.

7. **NO-SQL** (03 Periods)

Inroducton, Usages, and Application.

8. **SECURITY** (05 Periods)

Authorization and View- Security constraints - Integrity Constraints- Encryption

LIST OF PRACTICALS
STRUCTURED QUERY LANGUAGE

1. Creating Database
   - — Creating a database
   - — Creating a table
   - — Specifying relational data types
   - — Specifying constraints
   - — Creating indexes

2. Table and Record Handling
   - — INSERT statement
   - — Using SELECT and INSERT together
   - — DELETE, UPDATE, TRUNCATE Statement.
   - — DROP, ALTER statement

3. Retrieving Data From a Database
   The SELECT statement
   - — Using the WHERE clause
   - — Using Logical Operators in the WHERE clause
   - — Using In, BETWEEN, LIKE, ORDER BY, GROUP BY & HAVING clause
   - — Using Aggregate Functions
   - — Combining Tables Using JOINS

4. Design of database for any application.

Vertexal.

विषय-सूची

1

डाटाबेस सिस्टम और डाटा मॉडलिंग
(DATABASE SYSTEM CONCEPT AND DATA MODELLING)

1.1 सूचना प्रक्रिया (INFORMATION PROCESSING)

यह दी हुई या प्राप्त हुई सूचना को उपयोगी एवं सार्थक रूप में बदलने की प्रक्रिया है। सूचना को processed, organised (संगठित) और classified (वर्गीकृत) किया जाता है तकि वो user के काम आ सकें। Information एक प्रोसेस्ड डाटा (Processed data) है जो उसी रूप में या किसी और Data या Information के साथ इस्तेमाल किया जाता है। सूचना को प्राप्त करने वाला (Receiver) सूचना के आधार पर Action और Decision (निर्णय) लेता है। Collect किए गए डाटा से उपयोगी और अर्थपूर्ण सूचना निकालने के लिए उसे process किया जाता है, इस उपयोगी अर्थ को Information कहते हैं। Information (सूचना) को सही अर्थों में उपयोगी कहा जाता है अगर उसमें निम्न विशेषताएँ होती हैं–

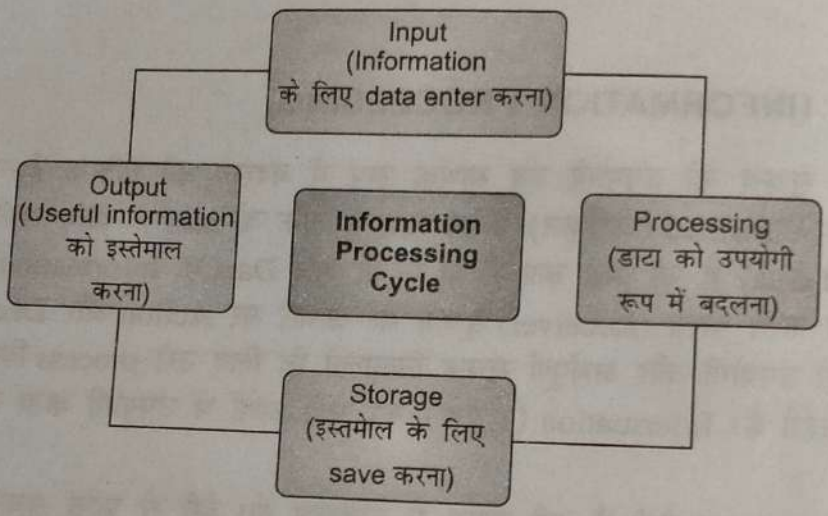
- **Timely:** वही सूचना उपयोगी है, जो समय से उपलब्ध हो। देरी से प्राप्त सूचना user के लिए useless (अनुपयोगी) होती हैं।
- **Accurate:** सूचना का सही यानि Accurate होना अति आवश्यक है। सूचना में छोटी-छोटी गलतियों की भी सम्भावना नहीं होनी चाहिए।
- **Completeness:** सूचना पूरी होनी चाहिए। आधी-अधूरी सूचना से गलतियाँ हो सकती हैं और अनचाहा परिणाम (Result) मिल सकता है।
- **Comprehensive:** वह सूचना जो व्यापक नहीं है user के लिए बेकार है। जब sender तो सूचना भेज देता है पर वह user के काम की नहीं है तो यह information failure है और ऐसी सूचना को 'सूचना' नहीं कहा जा सकता।

Information एक processed data है, जो किसी व्यक्ति के लिए कुछ उपयोगी होता है। Information में आगे भी process करके या कुछ सुधार करके उसे ज्ञानवर्धक बनाया जा सकता है।

Information processing के उदाहरण को हम Report card में दिए गए अंकों से समझ सकते हैं। यह एक अर्थपूर्ण और Processed Information है। इसी प्रकार तथ्यों से Report तैयार की जाए तो वो समझने में आसान होती है। सर्वेक्षण (survey) के दौरान इकट्ठा किए गए डाटा को अगर process न किया जाए तो वह अर्थहीन होता है। अगर सर्वेक्षण के दौरान formats बनाकर डाटा को format में collect करके process किया जाए तो यह उपयोगी होता है। इसी प्रकार अगर हम Table में store data को केवल Tabular form में देखें तो ज्यादा उपयोगी सूचना या संदेश नहीं मिलता लेकिन अगर उसी Tabular data को Graphs एवं Chart के रूप में इस्तेमाल किया जाए तो सही और स्पष्ट सूचना प्राप्त होती है। इसलिए Information को process करना बहुत ज़रूरी है।

सूचना प्रसंस्करण चक्र (Information Processing Cycle): इसमें मुख्य रूप से चार stages होते हैं: Input, processing, storage और output. Information Processing Cycle की ज्यादा जानकारी के लिए हमें पहले Data Processing Cycle को समझना जरूरी है। यह सभी stage Data Processing Cycle में भी समान होते हैं। उदाहरण के लिए अगर हमें कम्प्यूटर पर कुछ उपयोगी कार्य करना है तो, पहले उसे External source से Data या Instructions देनी पड़ती है। Computer Input Stage में Data/Instructions Receive करता है जैसे keyboard के द्वारा या डिस्क के द्वारा। Data पर Instruction/Commands को apply करने का काम Data processing की stage पर किया जाता है। यहाँ Data को उपयोगी रूप में बदल दिया जाता है और उसे इस्तेमाल के लिए save कर लिया जाता है। Information saving के इस stage को storage कहा जाता है। इसके बाद output stage में processed और इस्तेमाल के लायक data को use किया जाता है।

Information Process Cycle



Information Process Cycle के चार Stages होते हैं–

1. **Input:** कम्प्यूटर डाटा और निर्देशों को प्राप्त करता है।
2. **Process:** कम्प्यूटर निर्देशों के अनुसार उपयोगी Information प्राप्त करने के लिए डाटा को process करता है।
3. **Storage:** Information को आगे इस्तेमाल के लिए save करता है।
4. **Output:** Information को उपयोगी रूप में user को भेज देता है।

Input—कम्प्यूटर में data enter करना।

- इकट्ठे किए गए डाटा को processing के लिए cycle (चक्र) में डालना। यह unprocessed received data है जो सूचना प्राप्त करने के लिए process किया जाता है।
- Data को enter करने के लिए की-बोर्ड, माउस, स्कैनर, Discs, Joystick, डाटा टेबल (ड्रॉईंग्स) इत्यादि का इस्तेमाल किया जाता है।

Processing—डाटा पर काम करना (process करना)—Input देने के बाद दिए गए तरीके या निर्देशों से data को process किया जाता है। यह एक बहुत महत्त्वपूर्ण step है जो Processed data को output की form में करने के बाद इस्तेमाल किया जा सकता है।

कम्प्यूटर में processing CPU (Central Processing Unit) के द्वारा की जाती है। CPU कार्य करने के लिए एक बहुत आवश्यक component है।

Storage—डाटा को soft/physical रूप में save करना—यहाँ जो डाटा कम्प्यूटर में enter किया गया है उस डाटा को कम्प्यूटर में स्टोर किया जाता है। अब यह उपयोगी सूचना में बदल चुका है। डाटा स्टोर करने के लिए Hard Disk, Pen drive, Micro SD card, C D इत्यादि का उपयोग किया जाता है।

Output: यहाँ processed सूचना उपयोगी रूप में प्राप्त होती है। इस चरण (step) में processing के result को इकट्ठा किया जाता है। Output डाटा को कई तरह से इस्तेमाल किया जा सकता है।

1.2 डाटाबेस की शब्दावली (DATABASE TERMINOLOGY)

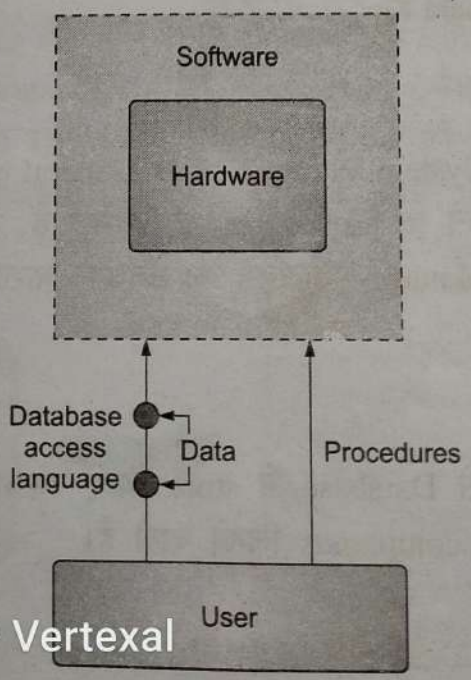
दिए गए डाटा को जब तक process नहीं किया जा सकता जब तक उसे एक दिए गए format में organise ना किया जाए, डाटा को characters, fields, records, files और databases के रूप में organise किया जाता है। इन शब्दों के मतलब हम आगे detail में समझेंगे। इनको समझने के बाद आप निम्न कार्य कर सकते हैं:

- Database को समझने और इस्तेमाल करने के लिए आवश्यक key terms को जानना।
- Database Processing में character के role और purpose (भूमिका और उद्देश्य) का पता करना।
- Fields के role और purpose का पता करना।
- Records के role और purpose का पता करना।
- Databse files के कार्य और इस्तेमाल के बारे में पता करना।
- Database के role और purpose को पता करना।
- Database management system के Role और purpose का पता करना।
- Data processing में key के महत्व का पता करना।

Character: Character Database में इस्तेमाल की जाने वाली सबसे basic unit होती है। एक character एक single symbol होता है, जैसे–Digit (0, 1, 2, 3, 4, 5, 6, ...) letter (a, b, c, ... etc.) या special character (जैसे $, #, ?). Character Bits और Bytes के रूप में कम्प्यूटर में store होता है।

1.3 DATABASE MANAGEMENT SYSTEM के घटक (COMPONENTS)

Database Management System को प्रमुख पाँच घटकों में बाँटा जा सकता है—



1. Hardware (हार्डवेयर)
2. Software (सॉफ्टवेयर)
3. Data (डाटा)
4. Procedures (प्रक्रिया)
5. Database Access Language

हार्डवेयर (Hardware)

कम्प्यूटर हार्डवेयर का मतलब भौतिक उपकरण से है। जैसे कम्प्यूटर, Hard disk, Input/output उपकरण जैसे keyboard, mouse, प्रिंटर इत्यादि। जब हम Database को कम्प्यूटर पर Run करते हैं तो उसकी Hard disk, Keyboard, ROM, RAM आदि Database के Hardware के Part होते हैं।

सॉफ्टवेयर (Software)

Software एक तरह का program है जो सब कुछ control करता है। DBMS (Data Base Management System) सॉफ्टवेयर कोड का एक संग्रह है जिसके द्वारा हम data को store, use और update कर सकते हैं।

DBMS software, Database Access Language को समझने में भी सक्षम होता है और उसे database commands में भी परिवर्तित कर देता है।

डाटा (Data)

Data वो resource है जिसके लिए DBMS को design किया गया है। DBMS को बनाने का उद्देश्य ही data को store और utilise करना है। एक typical database में user द्वारा save किया Data होता है और Meta data भी store होता है।

Meta data का मतलब data के बारे में data होता है। Metadata data base में store data के बारे में information देता है, उसे बेहतर समझने के लिए।

उदाहरण के लिए, जब हम My 'Name' को database में store करते हैं, तब DBMS अपने आप कुछ Information store कर लेता है, जैसे 'Name' कब store किया गया, 'Name' का size क्या है, क्या यह related या independent data है, यह सभी Informations metadata है।

प्रक्रियाएँ (Procedures)

Procedures Database Management System को use करने में General Instructions (सामान्य निर्देश) होते हैं। इसमें DBMS को setup और install करने के procedures भी शामिल हैं, जैसे DBMS software से login और logout करने के लिए, Database को Manage (डाटाबेस का प्रबंधन) करने के लिए, Backup लेने के लिए और Report generation के लिए।

Database Access Language

यह Database की सरल भाषा है जिससे Database में store किए गए data को access करने, insert करने, update करने और delete करने के लिए commands लिखी जाती हैं।

कोई user database access language में commands लिखकर DBMS में submit कर सकता है जिसे बाद DBMS द्वारा अनुवादित (translate) और कार्यान्वित (execute) किया जाता है।

User, access language का इस्तेमाल करके नया database, tables बना सकता है और डाटा को Insert करना, stored data को fetch/access करना, data को update/delete करना जैसे कार्य कर सकता है।

Users

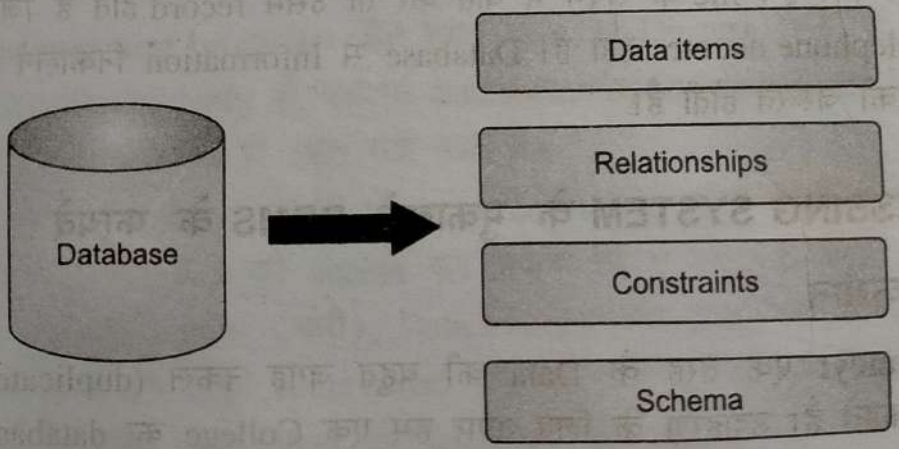
- **Database Administration:** Database Administrator या (DBA) वह व्यक्ति है जिसके पास पूरे Data की जिम्मेदारी होती है। जो पूरे DBMS का प्रबंधन करता है। उसे DBMS की Secuirty, availability, license key को manage करना, user accounts को Manage करना जैसी जिम्मेदारियों को निभाना पड़ता है।
- **Application Programmer or Software Developer:** यह User group DBMS के Parts को Design और Develop करने का कार्य करता है।
- **End user:** आजकल जितनी भी Modern application हैं, (web या mobile की) user data को store कर लेती है। आपने सोचा है ये application ऐसा कैसे करती हैं। ये applications इस तरह से programmed की जाती हैं कि वे user को data को collect करके DBMS system के server में store कर सकें। End user data को store, retreive (दोबारा से use करना) update और delete कर सकते हैं।

1.4 डाटा बेस के अवयव एवं उसका संगठन (COMPONENTS AND ORGANIZATION OF DATABASE)

एक database में निम्नलिखित चार elements होते हैं–

(*i*) Data
(*ii*) Relationships
(*iii*) Constraints
(*iv*) Schema

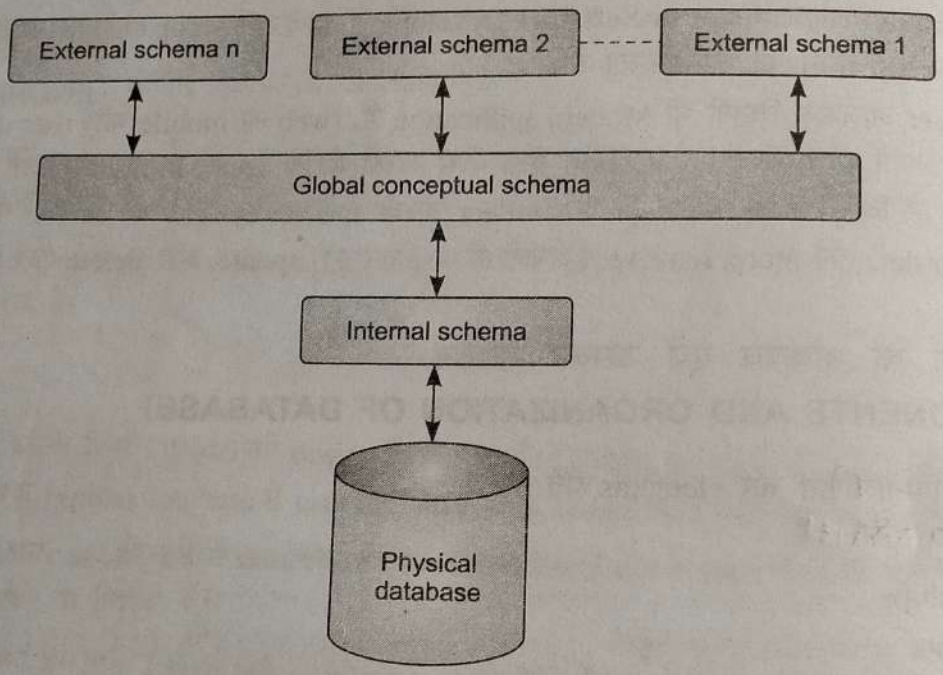
Data store की हुई logical entities का Binary representation होता है। विभिन्न तरह की सूचनाओं को special तरीके से format किया जाता है। जैसा हम जानते हैं computer software को दो श्रेणियों में विभाजित किया जाता है। Data और Program, जहाँ Progam Data को Manipulate (जैसा हम Result चाहते हैं) करने के लिए दी गई Instructions का collection है। DBMS में 'File' Database की Information को Store करती है और Index file और Data Dictionary वो Information store करती है, जिसे Meta data कहते हैं।



Relation विभिन्न प्रकार के Data elements के बीच संबंध को दर्शाती है। Constraint यह सुनिश्चित (ensure) करते हैं कि डाटा सही है। Schema, database के लिए एक organisation या structure है जो यह दर्शाता है कि database में data किस प्रकार organise है तथा उनके मध्य relationship कैसी है। यह data representation के physical और logical aspect को अलग-अलग करता है।

निम्न चित्र database के organisation (संगठन) को दर्शाता है। Internal schema बताता है कि डाटा Physical data storage में कैसे और कहाँ organised है। conceptual schema परिभाषित करती है। stored data का structure क्या है। External schema विशेष प्रकार के user या users के लिए है।

डाटाबेस, database access करने के लिए service provide करता है और साथ-साथ stored data की शुद्धता (accuracy) और consistency (निरंतरता) को भी बनाए रखता है।



विभिन्न प्रकार के लोगों द्वारा इस्तेमाल किए जाने के लिए design किए गए Data के collection को database कहते हैं। यह आपस में संबंधित Data का collection है, जिसे Redundancy (duplicacy) को control करने के लिए एक साथ store किया जाता है, एक या ज्यादा Applications को Serve करने के लिए इसका Use किया जाता है।

Database इस तरह संगठित (organised) किया जाता है कि computer program उससे desired data को Quickly select कर सके। केवल एक दी गई Information को Field कहते हैं और Field के Complete set को Records कहते हैं। अगर हम file के संदर्भ में बात करें तो उसमे record होते हैं जिसके अन्दर Fields जैसे कि Name, address, telephone no. etc. होते हैं। Database से Information निकालने के लिए हमें Database Management system की जरूरत होती है।

1.5 FILE PROCESSING SYSTEM के मुकाबले DBMS के फायदे

File System के नुकसान

- **Data redundancy:** एक तरह के Data की बहुत जगह नकल (duplicate data) को हम Data Redundancy कहते हैं। उदाहरण के लिए अगर हम एक College का database बना रहे हैं और एक

विद्यार्थी ने एक साथ दो courses में enroll (admission) किया है तो एक ही Details जैसे कि नाम, पता, फोन नं. को दो बार store किया हुआ हे, और हमें ज्यादा Storage space की जरूरत पड़ती है। ज्यादा और Repeated data होने से storage का मूल्य (cost) बढ़ जाता है और Data को access करने में ज्यादा समय लगता है।

- **Data inconsistency:** Data Redundancy से Data Inconsistency (Data विसंगति) भी बढ़ती है। ऊपर वाले college के उदाहरण को ही गर दोबारा देखें जहाँ दो course में enroll करने वाले विद्यार्थी का Data दो जगह store है, अब अगर student address को बदलने के लिए request करता है तो हो सकता है Address सिर्फ एक जगह change हो जाए और पूरे Records में ना हो क्योंकि Data अलग-अलग Files में है इससे Data Inconsistency (Data विसंगति) बढ़ती है।
- **Data Isolation:** File processing में Data विभिन्न Files में बिखरा होता है, और हर file का format अलग-अलग हो सकता है। इससे appropriate data को Retrive करने के लिए Application program लिखना मुश्किल होता है।
- **Dependency on application programs:** अगर files में कुछ बदलाव करना है तो application program में भी बदलाव करना पड़ेगा। file processing application program पर निर्भर करता है।
- **Atomicity issues:** किसी भी Transaction (लेन-देन) में atomicity का मतलब 'All or nothing' होता है मतलब या तो सभी operation execute होगें या एक भी नहीं। **उदाहरण के लिए:** किसी Bank transaction में रमेश ने सुरेश के account में 10000 रुपए transfer किए। अब computer system fail हो जाने के कारण पहली transaction तो हो गई रमेश के account से पैसे debit हो गए पर दूसरी transaction fail हो गई सुरेश के account में पैसे credit नहीं हुए। इस case में transaction को Atomicity को Maintain रखने के लिए पूरे operation का Reverse होना जरूरी है। जो कि File processing system में करना मुश्किल होता है।
- **Data Security:** Data की हमेशा unauthorized access (गलत इस्तेमाल) से सुरक्षा होनी चाहिए जैसे कि college के student को teachers का payroll देखने का authorizsation/access नहीं होना चाहिए। इस तरह के security constraint को file system में लागू करना मुश्किल होता है।

Advantage of DBMS Over File System (DBMS के फायदे)

DBMS के File processing system के ऊपर बहुत से फायदे हैं जैसे कि–

- **No redundant data:** Data की duplicacy (Rebundancy) को Data normalization से Remove किया जा सकता है। इससे Duplicate data storage की जगह नहीं लेता और access time भी improve (कम लगता है) होता है।
- **Data Consistency and Integrity:** जैसे हमने पहले भी Discuss किया है Data inconsistency का मुख्य कारण Data Redundancy है, यहाँ पर data normalisation data Redundancy नहीं होने देता, Data inconsistency का भी उससे ही ध्यान रखा जाता है।
- **Data Security:** Datanbase system में access constraints लागू करना आसान होता है, इससे अधिकृत (authorised) user ही Data को access कर सकता है। प्रत्येक user के लिए access का Different set होता है, इससे Data theft (चोरी), Data leakage और Data के गलत इस्तेमाल से सुरक्षा मिलती है। डाटाबेस में किसी प्रकार की value insert करने से पहले उसकी कुछ conditions को satisfy करना आवश्यक होता है। इससे Data safety और Integrity बढ़ती है।

- **Privacy:** Limited access (सीमित पहुँच) से Data की Privacy बढ़ती है।
- **Easy access to data:** Database system डाटा को इस तरह व्यवस्थित करता है की Data को आसानी से access किया जा सके और कम से कम समय में भी access किया जा सके।
- **Easy recovery:** चूँकि Database system Data का Backup रखता है, इसलिए Data Failure के समय में Data की Full recovery आसान हो जाती है।
- **Flexible:** Database File processing system से ज्यादा Flexible (लचीले) होते हैं।

Disadvantages of DBMS (DBMS की हानियाँ)

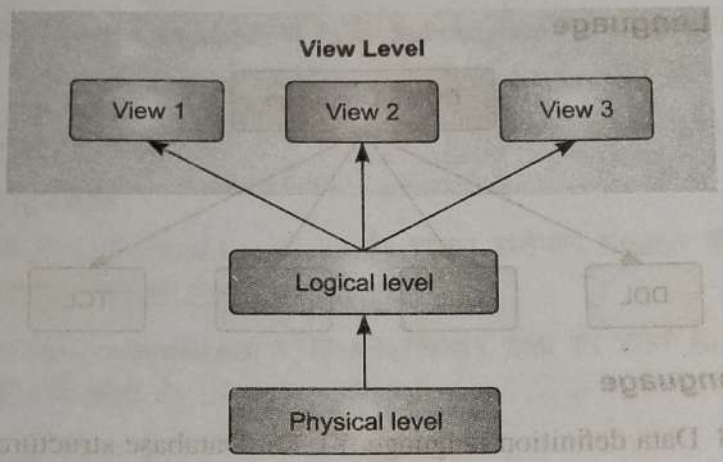
- DBMS को Implement करने का मूल्य File system से बहुत ज्यादा होता है।
- **Complexity:** (जटिलता) DBMS की Functionality जटिल होती है। Proper Training देने की जरूरत होती है।
- **Updation:** इसको लगातार Upadate करना पड़ता है जो बहुत कठिन होता है।

1.6 DATA ABSTRACTION AND DATA INDEPENDENCE

Database system complex data-structures से मिलकर बना होता है। System को Data retrieval के लिए efficient (प्रभावशाली) बनाने के लिए, users के लिए आसान बनाने के लिए, Developers abstraction का इस्तेमाल करते हैं, इसका मतलब है बिना काम की जानकारी (irrelevent details) को user के लिए hide कर देना। यह approach डाटाबेस के design को आसान बना देती है।

Data Abstraction मुख्य रूप से तीन प्रकार की होती है:

- **Physical:** यह abstraction का सबसे निचला स्तर (level) है। ये हमें बताता है कि Data को वास्तव में Memory में किस प्रकार Store किया गया है। इसके लिए access Methods जैसे कि sequential या Randaom access और file organisation methods जैसे कि B+ trees, hasing का उपयोग किया जाता है। डाटा की उपयोगिता, Memory का size, Records का कितनी बार use करना ऐसे factors है जिनका Database को Design करते समय जानना जरूरी है।
- **Logical:** यह level वो Information बताता है जो Database में Tables के रूप में stored होती है। यह Data entities के मध्य के Relationship को भी simple strcutures के रूप में store करके रखता है। यह level बताता करता है कि DBMS में क्या Data है और उनके मध्य क्या Relationship है।
- **View:** यह abstraction का सबसे highest level है। ज्यादातर user पूरे database का प्रयोग नहीं करते और उनका कुछ भाग ही read करते हैं। यह Individual user के लिए database के उसी भाग को दिखाता है। User Data को Row या Columns के Format में देख सकते हैं। Tables और relation को Data को Store करने के लिए इस्तेमाल किया जाता है। एक डाटा की कई तरीके से भी देखने की सुविधा इस level में मिलती है। (डाटा के विभिन्न view हो सकते हैं।) User को केवल Data को View करके ही Database से Interact कर सकता है, storage और implementation की जानकारी उससे छुपी हुई होती है।



Abstraction का मुख्य उद्देश्य Data Independence को achieve करना है ताकि जब Database को Modify या alter (बदलाव) करना हो तो समय और cost की बचत हो।

Data Independence के मुख्य रूप से दो प्रकार हैं:

- **Physical Level Data Independence:** Logical schema या conceptual schema में बदलाव किए बिना physical schema में बदलाव करना Physical data Independence कहलाता है, इसे optimization के लिए किया जाता है, उदाहरण के लिए अगर Database के system server के storage size में कोई बदलाव किया जाए तो, conceptual structure में कोई बदलाव नहीं होता। Sequential से Random access files में बदलाव भी इसका एक उदाहरण है।

  Physical structure (भौतिक संरचना) में बदलाव या संशोधन निम्न प्रकार का हो सकता है।

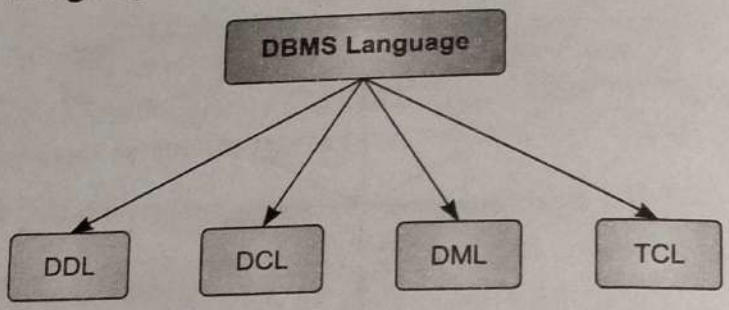
  - नए Storage devices का उपयोग
  - Storage के लिए use किए गए Data structure में संशोधन (Modification)।
  - Indexes में बदलाव या File organisation की alternative (वैकल्पिक) तकनीक का इस्तेमाल करना।

- **Logical Level Data Independence:** बाहरी Schema या application Program में बदलाव किए बिना Logical Schema को बदलने में सक्षम होने की विशेषता को Logical data Independence कहते हैं। यदि हम Data के Conceptual view में कोई भी परिवर्तन करते हैं तो Data का User view प्रभावित नहीं होगा। इन बदलावों में attributes का addition या Deletion, table structure entities में बदलाव या Logical Schema में Relationships में बदलाव शामिल हैं।

1.7 DATABASE LANGUAGES

- किसी System में Database को Create और Maintain करने के लिए database langauges का प्रयोग किया जाता है।
- DBMS language को Database के Data को read, store या update करने के लिए किया जाता है।
- Database langauage का इस्तेमाल queries या updates को express करने के लिए भी किया जाता है।

Types of database Language



1. Data Definition Language

- **DDL** का पूरा नाम Data definition language है। इसे Database structure या Pattern को Define करने के लिए इस्तेमाल किया जाता है।
- इसे Database में Schema, tables, Indexes, Constraints इत्यादि को Create करने के लिए इस्तेमाल किया जाता है।
- DDL Statements से Database के ढाँचे (skeleton) को तैयार किया जा सकता है।
- DDL का इस्तेमाल Metadata की Information जैसे table और schema की संख्या, उनके नाम, प्रत्येक table के columns और indexes, constraints इत्यादि को store करने के लिए किया जाता है।

कुछ महत्त्वपूर्ण DDL statements को नीचे दिया गया है–

- **Create:** डेटाबेस में Objects को Create करने के लिए इस्तेमाल की जाती है।
- **Alter:** डेटाबेस के Structure में बदलाव के लिए इस्तेमाल किया जाता है।
- **Drop:** डेटाबेस में से Objects को Delete करने के लिए इस्तेमाल किया जाता है।
- **Truncate:** किसी Table में से सारे Records को Delete करने के लिए।
- **Rename:** किसी Object का नाम बदलने के लिए।
- **Comment:** Data dictionary में Comment को add करने के लिए।

ये सभी Commands डेटाबेस Schema को Update करती है। इसलिए DDL के अंतर्गत आती हैं।

2. Data Manipulation Language

DML का पूरा नाम Data Manipulation Language है, ये डेटाबेस में Data को access और Manipulate करने के काम आती है। ये user requests को भी handle करती है। कुछ महत्त्वपूर्ण DML Statements/commands नीचे दी गई हैं–

- **Select:** एक डेटाबेस में से Data को Retrive करना।
- **Insert:** Table में डेटा को insert करना।
- **Update:** Table में मौजूद डेटा को Update करना।
- **Delete:** Table में से सभी Records को Delete करना।
- **Merge:** ये UPSERT Operation करती है, मतलब एक साथ Insert और Update दोनों करना।

- **Call:** Structured Query Language या Java Subprogram को Call करने के लिए।
- **Explain Plan:** Data को Explain करने के लिए।
- **Lock Table:** Concurrency (एक साथ दो घटनाएँ होना) को नियंत्रित करने के लिए।

3. Data Control Language

- **DCL** का मतलब Data Control Language है। इसका इस्तेमाल Stored या Saved data के retrieval (access) को नियंत्रण करने के लिए किया जाता है।
- DCL का Execution transactional (लेन-देन संबंधी) होता है। इसमें Roll back parameter (वापिस करने का प्रावधान) भी होता है।

मुख्य DCL Commands निम्न हैं–

- **Grant:** Database के लिए Users को Privilege (विशेषाधिकार) प्रदान करने के लिए।
- **Revoke:** User से Permission (अधिकार) वापिस लेने के लिए।

नीचे दिए गए Operations के लिए भी Revoke का प्रावधान होता है।

CONNECT, INSER, USAGE, EXECUTE, DELETE, UPDATE and SELECT

4. Transaction Control Language

TCL का उपयोग DML Statement द्वारा किए गए Changes (बदलाव) को RUN करने के लिए किया जाता है। TCL को कुछ Logical transaction में समूहित (Grouped) किया जाता है। यहाँ कुछ tasks हैं जो TCL के अन्तर्गत आते हैं–

- **Commit:** इसे Database में transaction को save करने के लिए किया जाता है।
- **Rollback:** यह commit की विपरीत command है, इसे commit के बाद हुए बदलाव को restore करने के लिए किया जाता है।

1.8 DATABASE BASICS

क्योंकि किसी भी Organisation (संस्था) के Database से Organisation में की जाने वाली activities के बारे में पता चलना चाहिए, database के basic concept, principles और इसमें use की जाने वाली terms (शब्दावली) के बारे में जानना बहुत आवश्यक है।

Data-Items

Data Processing में Data item का मतलब वो term है जिसे पारम्परिक रूप से 'Field' कहा जाता है और यह data की सबसे छोटी इकाई (smallest unit) है जिसका User के लिए कुछ meaning होता है। Data item की जगह हम कभी-कभी data-element या elementary item जैसे शब्द भी प्रयोग कर सकते हैं। Data item को database का molecule (एक कण) कहा जा सकता है, जो मिलकर कुछ समूह बनाते हैं जिन्हें विभिन्न नामों से जाना जाता है। उदाहरण के लिए Data items के एक समूह को Records कहा जाता है और एक program सभी

Records को read करता है। Data Items को कभी-कभी आगे भी तोड़ा जाता है जिसे Processing Purpose के लिए automatic level कहते हैं।

Entities and Attributes

Real world में कुछ tangible (वास्तविक) objects हाते हैं जैसे कि कोई employee; Inventory का कोई component या space या फिर इसमें कुछ Intangible objects होते हैं जैसे कि Job description; identification no.,; इत्यादि। ऐसे सभी Items जिनके बारे में Information database में store होती हैं entities कहलाते हैं। किसी entity की विशेषताएँ (properties) जिन्हें हम Information के रूप में store करते हैं, attributes कहलाती हैं, वो entity के बारे में सूचना provide करती है। Attributes कुछ भी हो सकता है जैसे कि कोई no या text, या कोई scanned picture या sound sequence या moving pictures जो आजकल कुछ Multimedia-database में इस्तेमाल की जा सकती है।

उदाहरण से समझें: अगर Student एक entity है तो उसकी properties जैसे name, class, age, Roll No. उसके attributes हैं।

साधारणतया: database processing भी एक तरह की entities का collection होता है और entities के एक तरह के attributes के बारे में Information record की जाती है। परंपरागत approach में programmer प्रत्येक entity का Record maintain करते थे और Records के data item प्रत्येक attribute से सम्बंधित होते थे। इसी प्रकार Records, files में समूहित (grouped) किए जाते थे।

Logical and Physical Data

Database का एक प्रमुख feature यह है कि ये data के logical और physical structure को अलग-अलग करता है। Logical structure का मतलब है कि programmer data को कैसे देखता है और physical strucutre का मतलब कि Data storage medium में वास्तव में किस तरह store होता है। यहाँ तक की पुराने समय में जब records को tape में store किया जाता था, बहुत सारे logical records को इकट्ठा (group) करके एक Physical record में रखा जाता था। यह एक software होता था, जो इस्तेमाल के वक्त उन्हें अलग-अलग कर देता था और tape में store करते वक्त उन्हें वापस एक कर देता था। आजकल के वक्त में जटिलताएँ बढ़ गई हैं, database अलग-अलग और दूर-दूर जगह पर होते हैं।

Schema and Subschema

Schema database के लिए एक organisation या structure है जो database के logical view को represent करता है। Schema को chart के रूप में draw किया जाता है जिसमें इस्तेमाल हुए Data types का विवरण होता है। इसमें entities के नाम, उनके attributes उनके बीच के सम्बन्ध (Relation) की details होती है। उदाहरण के लिए airport या Railway station पर Information display system से आने और जाने का समय पता चलता है, जिसमें values समय-समय पर बदलती रहती है परन्तु Schema same ही रहता है। Schema की entities के बीच में सम्बन्ध (Relationships) कई तरह के हो सकते हैं जैसे कि one to one, one to many, many to many or conditioned.

Schema एक प्रकार से database में store किए गए सभी Data item types और record-types का overall chart है। (पूरे Data item का मानचित्र)।

Sub sachema, schema का ही एक छोटा रूप है। Sub schema का view schema की तरह ही होता है परन्तु Data items और Record type एक विशेष user द्वारा एक विशेष application में इस्तेमाल किए जाते हैं। एक Schema बहुत सारी sub schema से मिलकर बनी होती है।

Data Dictionary

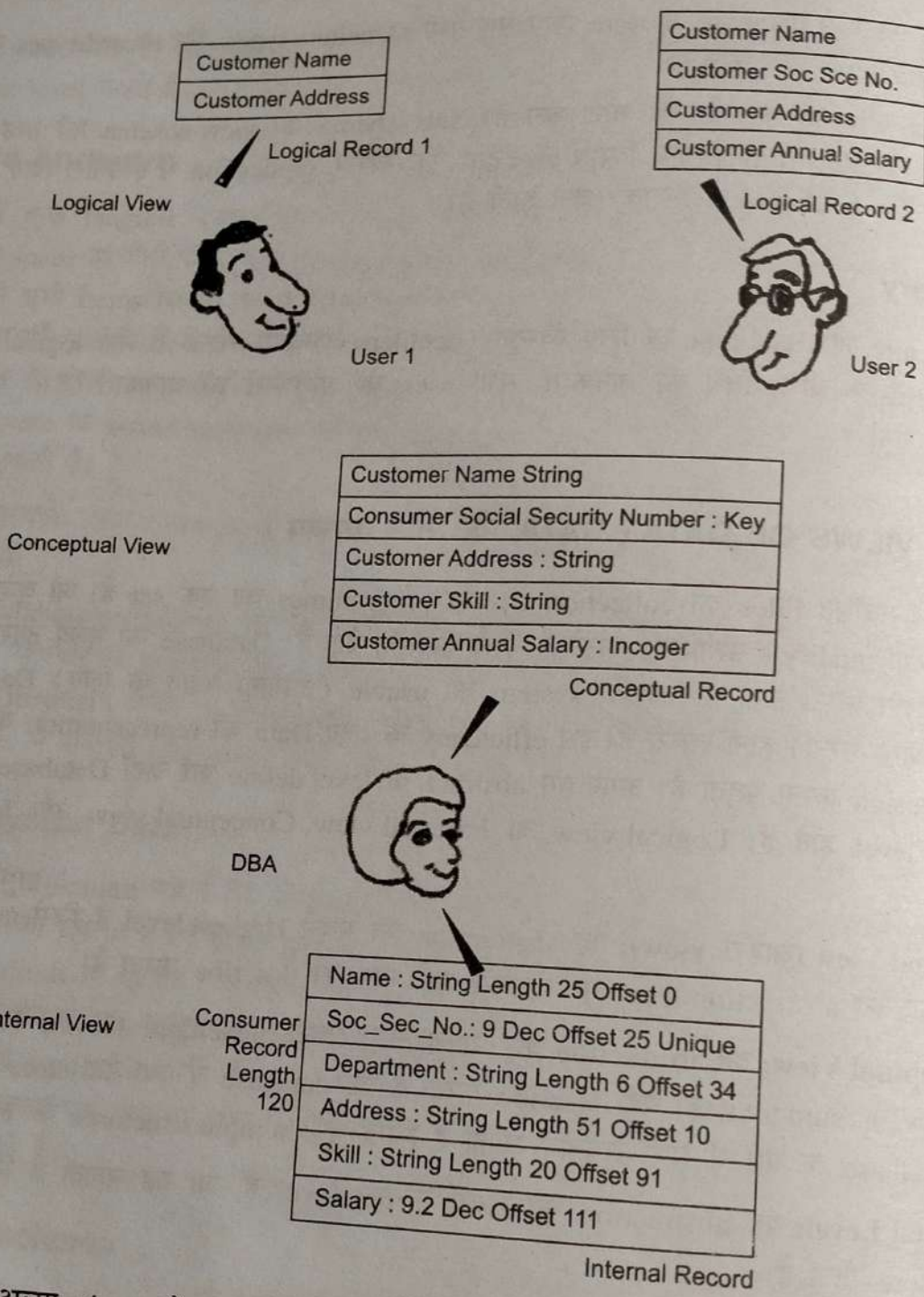
यह विभिन्न structure और data type के लिए विस्तृत (detailed) जानकारी रखती है जैसे logical structure की जानकारी, data items के बीच संबंध की जानकारी, सभी users के अधिकारों की जानकारी किसी भी Resources (संसाधन) का प्रदर्शन।

1.9 THREE VIEWS OF DATA (DATA के तीन प्रारूप)

DBMS आपस में संबंधित Files का collection है, और programmes का एक set है। जो कुछ users को इन files को access या modify (संशोधित) करने के लिए अनुमति देता है। Database का मुख्य उद्देश्य user को एक abstract view प्रदान करना होता है। तथापि, system को usable (उपयोगी बनाने के लिए) Data कुशलतापूर्वक Retreive (पुन: प्राप्त करना) होना जरूरी है। इस efficiency के लिए Data की representation के लिए complex data structure design करना पड़ता है। अगर हम abstract के level define करें जहाँ Database को view किया जाता है तो तीन level होते हैं: Logical view या External view, Conceptual view और Internal view या Physical view.

- **External View (बाहरी view):** यह abstraction का सबसे Highest level है जिसे user देख सकता है। इस प्रकार का abstraction level पूरे डेटाबेस एक भाग को describe करता है।
- **Conceptual View:** यह abstraction का अगला ऊँचा लेवल (next highest level) है, यह DBMS users के view का sum total है। यह describe करता है कि database में क्या डेटा store रहता है। यह level पूरे Database के बारे में सारी सूचनाएँ रखता है जिन्हें कुछ simple structures के रूप में रखा जाता है।
- **Internal Level:** यह abstraction का सबसे निचला level है, जो यह बताता है कि data physically कैसे store रहता है।

तीनों levels के बीच में सम्बन्ध को अगले पेज पर दिखाया गया है।



Data के अलग-अलग views में अन्तर जानने के लिए इसे programming languages के Data types के साथ compare किया जा सकता है। अधिकतर high level programming languages जैसे, C, C++ इत्यादि record या structure type की धारणाओं पर काम करती हैं। उदाहरण के लिए 'C' Language में Structure (Record) को नीचे दिए गए तरीके से declare किया जाता है।

```
struct
Emp
{
   char name [30];
   char address [100];
}
```

यह एक नए Record को जिसका नाम Emp है और उसके दो Fields हैं– name और address को Define करता है। हर field के साथ उसका नाम और Data type associate (जुड़ा) होता है।

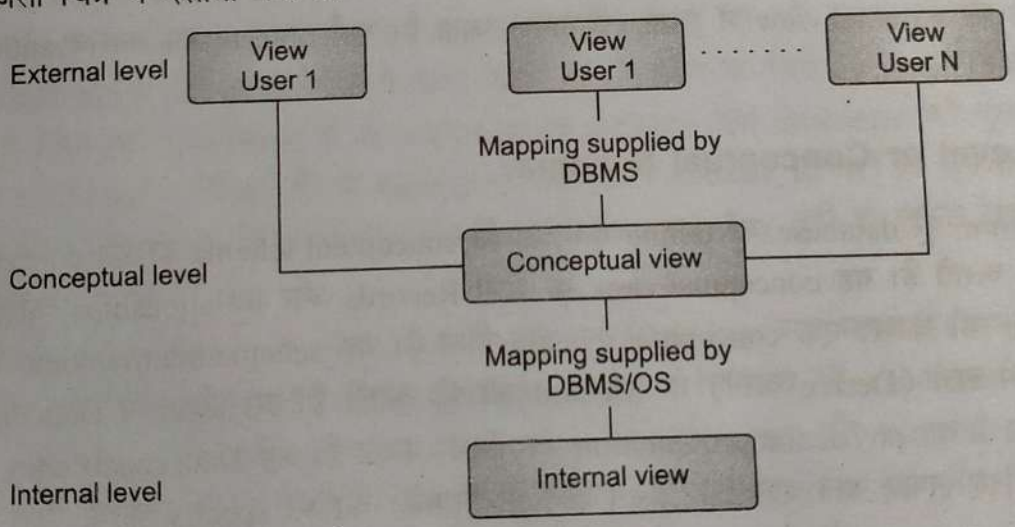
उदाहरण के लिए किसी बीमा कंपनी में इस तरह के विभिन्न प्रकार के records types होते हैं।

- Customer (Record) और उसके fields जैसे name and salary
- Premium Paid (कोई और Record है) और उसके Field हैं Due amount और Due date.
- Insurance agent का नाम (ये एक Record है) और उसकी Salary या commission उसके fields हैं।

Internal level में customer, उसका Premium account, employee बीमा एजेंट को describe किया जाता है। निरंतर Bytes के क्रम में conceptual level में प्रत्येक record को type definition से describe किया जाता है और records के बीच के सम्बन्ध को define किया जाता है। किस customer या end user को क्या-क्या rights (अधिकार) या सुविधाएँ प्रदान करती है। वो भी इस Level में describe किया जाता है। External level में हम, कई तरह के views को define कर सकते हैं। उदाहरण के लिए, Insurance के customer detail बनाते वक्त customer की details की ही जरूरत होती है ना कि उनके accounts की detail की इसी तरह teller (बैंक या कंपनी का गण, गिनती करने वाला) को सिर्फ account की Information का access चाहिए। इसलिए उन्हें कितना Premium pay हुआ या कितना amount Recieve हुआ उसका access नहीं होना चाहिए।

1.10 THE THREE LEVEL ARCHITECTURE OF DBMS

वो डाटाबेस Management system जो तीन level का Data provide करते हैं, वो three-level architecture का पालन करते हैं, जैसा चित्र में दर्शाया गया है।



Schema इन तीनों levels के view को describe करता है। जैसा पहले भी बताया गया है, schema एक outline या plan है किसी view के records और उनके बीच के सम्बन्ध (Relationship) को describe करता है। schema यह भी बताता है कि abstraction के एक level की entities को अगले level से कैसे map किया जाए। Database Schema किसी भी database का overall design है।

Database Schema में अग्रलिखित सूचनाएँ होती हैं–

Data item की विशेषताएँ (characterstics), जैसे की उनकी entities और attributes, storage representation का प्रारूप (Format), Integrity के लिए मापदंड (Parameters) जैसे की Physcially authorisation और backup politics. Data itmes का logical structure और उनके बीच सम्बन्ध।

Data base schema का Concept Programmaing language की 'type defination' की धारणा से मेल खाता है। किसी भी type के variable की एक दिए गए समय पर particular value होती है। Programming languages का variable की value वाला concept database schema के Instance के concept से मेला खाता है। चूंकि प्रतयेक view schema द्वारा defined होता है, database में कई तरह की schema होती है और ये data abstraction के तीनों levels को अलग-अलग करती है। सबसे निचले level पर physical schema होती है, बीच के level पर conceptual schema होती है और Highest (सबसे ऊपर) के Level पर Subschema होती है। सामान्य रूप से, database system: एक Physcial schema, एक conceptual schema और कुछ sub-schema को support करता है।

External Level or Subschema (बाहरी Level या Subschema)

बाहरी level database का उच्च level है, जहाँ database का एक भाग किसी विशेष user या application program के लिए define होता है। एक दिए गए conceptual या global view के लिए कितने भी user view (external level) हो सकते हैं।

प्रत्येक External view एक schema द्वारा describe किया जाता है जिसे Externa Schema या Subschema कहते हैं। External Schema में, Logical records की परिभाषा और external view में उनके बिच में सम्बन्धों का विवरण होता है। External schema में वह Method (तरीका) भी defined होता है जिसके द्वारा Conceptual\ view के objects को external view में लाया (दिखाया) जाता है। यहाँ objects का मतलब entities, attributes और relationship है।

Conceptual Level or Conceptual Schema

एक Conceptual view पूरे database को define कर देता है। conceptual schema ही इस conceptual view को परिभाषित (define) करती है। यह conceptual view के सभी Records और Relationships को describe करती है। प्रत्येक database की केवल एक conceptual schema होती है। यह schema internal view के objects को conceptual view में लाने (Derive करने) के Method को भी बताती है। इस level में Data का Description जिस format में होता है वह physical representation पर निर्भर करता है। यह Data consistency और Integrity को बनाए रखने के लिए लगाए जाने वाले checks (जाँच के नियमों) को भी specify करता है।

Internal (आन्तरिक) Level या Physical Schema

यह बताता है कि Data को किस प्रकार Store किया गया है, और यह database द्वारा इस्तेमाल किए गए data structuer और access method को भी describe करता है।

Internal scheme में, स्टोर किए गए Records की defintion होती है, data fields को represent करने का तरीका होता है। यह Internal view को व्यक्त करता है। और अगर कुछ access aids (अतिरिक्त तरीके) इस्तेमाल हुए हैं तो उनके बारे में भी बताता है।

विभिन्न Levels के बीच की Mapping

किसी भी Database system में जिसमें तीन views के साथ, दो तरह की mapping की जरूरत होती है। External और conceptual level के बीच की Mapping, external और conceptual levels के Records और Relationships के बीच के सामन्जस्य (आपसी सम्बन्ध) को बताती है।

(a) **External to conceptual:** यह निर्धारित करती है कि conceptual records user के द्वारा किस प्रकार देखे जाएंगे।

(b) **Internal to conceptual:** यह conceptual और Internal level के बीच व्यवहार स्थापित करती है। यह बताती है कि conceptual records को storage में कैसे दिखाया जाए।

एक Internal record, Internal level का एक Record है, जरूरी नहीं है कि वह physcial storage devices पर store किया गया कोई Record ही हो।

Internal record को दो या उससें ज्यादा Physcial records में बांटा जा सकता है। Physical database वह डाटा है जो Secondary storage devices में store किया जाता है। यह कुछ Data structures के साथ बनाए गए Records से मिलकर बनता है और files में organise किया जाता है। इसके फलस्वरूप Internal records और Secondary storage devices में store records के बीच भी एक अतिरिक्त mapping करानी पड़ती है।

- **Keys:** कोई key हमें किन्हीं attributes के set को Identify करने के लिए अनमुति प्रदान करती है जो कि entities को एक दूसरे से अलग करने के लिए पर्याप्त होते हैं। यह संबंधों (relationships) को भी एक-दूसरे से अलग करती है।
- **Super Key:** Super key एक या ज्यादा attributes का set है, जो हमें किसी भी entity के sets में से एक unique entity को अलग करने में मदद करता है। उदाहरण के लिए customer एक entity का set है जिसमें बहुत सारे customer हैं तो customer_id attribute एक customer को दूसरे से अलग करने के लिए sufficient (पर्याप्त) है। दो customers की एक customer_id नहीं हो सकती। इसलिए यहाँ पर customer id super key है। customer-name attribute, super key नहीं हो सकता क्योंकि कई लोगों के नाम एक जैसे हो सकते हैं।
- **Candidate key:** Super key के subset को हम candidate key कह सकते हैं। सभी candidate keys super key होती है। कुछ अलग-अलग attributes का set मिलकरा भी candidate key हो सकता है। उदाहरण के लिए अगर customer entity set में customer name और customer street, entity के सदस्यों को अलग कर सकता है तो वो candidate key है। यहाँ {customer_id} और {Customer_name, Customer street} दोनों candidate key हैं। हालांकि customer_id और customer_name भी मिलकर customer entities को अलग (distinguish) कर सकते हैं परन्तु उनका combination candidate key नहीं है क्योंकि customer_id खुद ही candidate key है और customer_name और customer_street अपने आप में candidate key नहीं हैं परन्तु मिलकर candidate keys बनते हैं।
- **Primary key:** वो key जो किसी table के प्रत्येक record को uniquely identify कर सकती है, primary key कहलाती है। Primary key database designer द्वारा candidate keys में से चुनी जाती है और यह किसी entity set में से एक entity को identify करने का मुख्य साधन है। कोई भी key (Primary, Candidate या Super key) किसी भी entity set की property होती है न कि किसी Individual entity

की। किसी भी set की दो अलग-अलग entities की key attirbutes की value समान नहीं होनी चाहिए। यदि हम ऐसा करने की कोशिश करते हैं तो भी DBMS हमें ऐसा नहीं करने देता क्योंकि वो, Primary key के constraint को follow करता है जिसके हिसाब से एक column में Duplicate value store नहीं हो सकती।

1.11 DBMS ARCHITECTURE

DBMS Architecture, database को design, develop (विकसित), Implement (लागू करना) और Maintain करने में मदद करता है। एक database business की बहुत महत्वपूर्ण सूचनाओं का store रखता है। इसलिए सही Database architecture को चुनने से Data के quick और secure access (जल्दी और सुरक्षित access) में मदद मिलती है।

1-Tier Architecture

सबसे Simple database architecture 1 tier है, जिसमें client, server और database एक ही machine में होते हैं। आप कभी भी database को अपने system पर Install कर सकते हैं, और SQL querries की practice के लिए इस्तेमाल कर सकते हैं।

यह 1-tier architecture होता है, परन्तु Production वाली companies में इसका इस्तेमाल कम ही होता है।

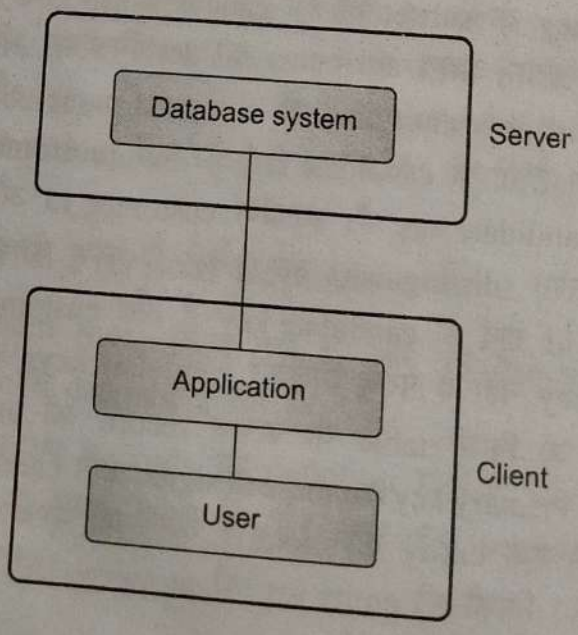
2-Tier Architecture

2-Tier architecture में निम्न Characterstics होती हैं–

- Presentration layer client पर Run की जाती है। (जैसे कि कम्प्यूटर, Mobile या Tablet पर)
- Data को Server पर Store किया जाता है।

एक application Interface जिसे ODBC (Open Database Conncetivity) कहा जाता है वह client side के program को DBMS को use करने की अनुमति देता है। आजकल अधिकतर DBMS, ODBC drives को ही इस्तेमाल करते हैं। 2-tier architecture DBMS को अतिरिक्त सुरक्षा प्रदान करता है, क्योंकि DBMS directly end user को exposed नहीं होता (दिखाई नहीं देता)। DBMS का Data अलग Server पर होता है।

उदाहरण के लिए MS-access से बनाया गया 2-tier architecture नीचे दिखाया गया है जो Contact Management Stystem भी कहलाता है।



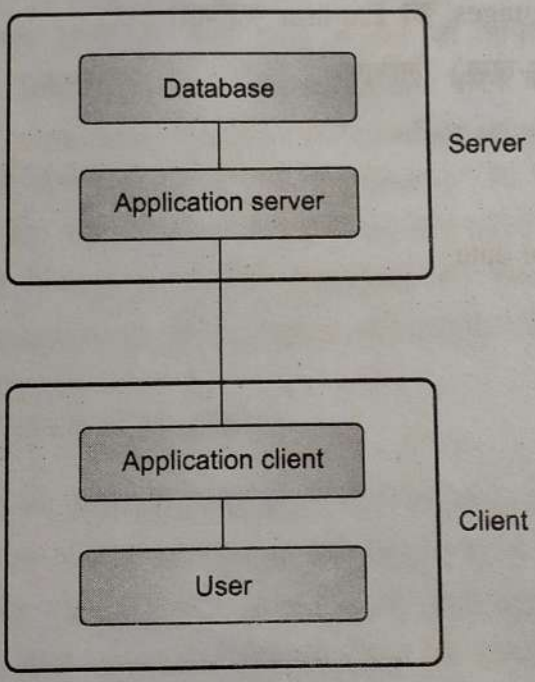
इसके अलावा एक 3-tier architecture भी होता है, जो 2-tier का एक extension है। इसके निम्नलिखित 3 layer होते हैं।

1. Presentation layer (client tier) (PC, Tablet, Mobile इत्यादि)
2. Application layer (server)
3. Database Server

इस DBMS structure में User or DBMS के बीच एक Application layer होता है, Application का काम है कि user की जरूरत को DBMS तक पहुँचाए और समझाए, और फिर वपिस DBMS के Response को User तक पहुँचाए (Communicate करे)।

Application Logic layer (या Business logic layer) user को या DBMS को Data पहुँचाने से पहले functional logic, constraints और Rules को भी Process करता है।

3-tier architecture सबसे प्रसिद्ध DBMS architecture है।



3-tier architecture के उद्देश्य नीचे दिए गए हैं–

- User application और Physical database को अलग अलग करना।
- DBMS की विशेषताओं को Support करना।
- Program-data की स्वायत्ता (Independence)
- Data के Multiple views को support करना।

3-tier Architecture का उदाहरण है–

इन्टरनेट पर कोई भी बड़ी website

अभ्यास प्रश्न

1. Data Management System के बारे में बताइए।
2. DBMS और File management के बीच के अन्तर को समझाइए।
3. Data Management System के क्या-क्या फायदे हैं?
4. विभिन्न प्रकार की Database terminology को बताइए।
5. DBMS के 3-level architecture के बारे में समझाइए।
6. Data और Information (सूचना) के बीच में क्या अन्तर है?
7. Database के अवयव (Components) क्या-क्या हैं?
8. Database की Organisation (संगठन) क्या है?
9. Database Management System के Components क्या-क्या होते हैं?
10. DBMS की विभिन्न Languages को Explain कीजिए।
11. निम्न पर Short Note (लघु-लेख) लिखिए।
    (क) Schema
    (ख) Data dictionary
    (ग) Logical and Physical data
    (घ) Entities
    (ङ) Conceptual view

2

डाटा मॉडल
(DATA MODEL)

2.1 परिचय (INTRODUCTION)

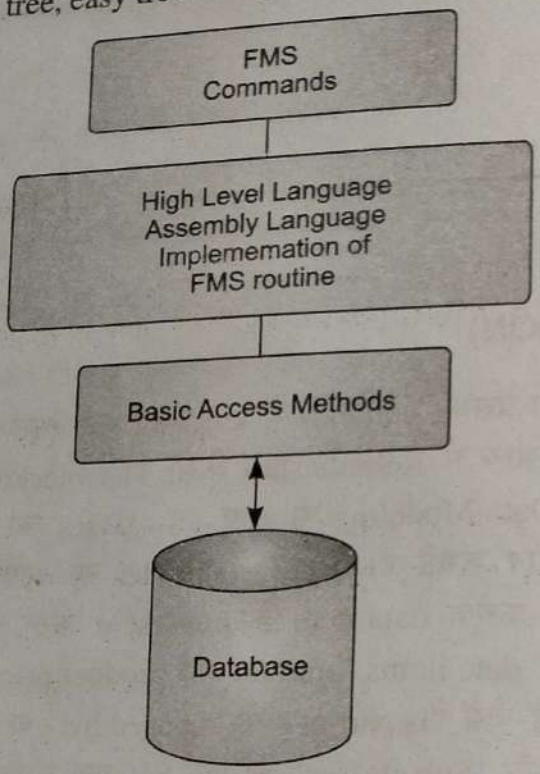
Data Modeling, Business के अन्य प्रसंगों (context) में use किए जाने वाले Data objects का विश्लेषण है, और यह बताता है कि Data objects के मध्य में Relationship कैसी है। Object orient programming करने के लिए Data Modeling पहला कदम है। Data Modeling से आप उन classes को define कर सकते हैं जो program के objects के लिए Template प्रदान करती हे। ऐसा Data model जो आपको Model को visualize करने दे, उसे बनाने का असान तारीका है कि प्रत्येक data item को represent करने के लिए आप एक square (या कोई और symbol) draw कर लें। जैसे data items (product और product price) के बीच सम्बन्ध दिखाने के लिए आप ऐसे शब्द इस्तेमाल कर सकते हैं जैसे "is part of" या "is used by" या "uses" इत्यादि। इस तरह के विवरण (Description) से आप 'Classes' और 'Sub classes' का एक Set बना सकते हैं, जो सभी General relationships को Define करता है। और यही बाद में objects के लिए Template का काम करता है और जब ये Program के रूप में execute होता है तो new transactions के variables को प्रभावशाली तरीके से handle करता है।

2.2 FILE MANAGEMENT SYSTEM (FMS)

Data processing के शुरूआती दिनों में, सभी files flat files होती थी। Flat file में प्रत्येक Record में Same type के data item होते हैं। इनमें से एक या अधिक data items को 'Key' बनाया जाता है, और File की Sequencing के लिए इस्तेमाल किया जाता है, और (Sorting और Indexing के द्वारा) records को locate और Group करने के लिए किया जाता है। इस तरह के सभी structures को बंद किया जा सकता है। ये 'tree' या 'clause structure के तरह होते हैं। फिर भी यह हमेशा ध्यान रखना चाहिए कि इस सभी तरह के complicated file structures को flat files के groups में बदला (break) जा सकता है जिनमें Redundant (एक तरह के) data items होते हैं।

एक FMS में बहुत सारे application programmes होते हैं। क्योंकि FMS को use करने पर एक conventional high level language, के मुकाबले productivity enhancement लगभग 10 : 1 होता है, इसलिए programmes इसे use करते हैं। FMS का use उन end users को encourage करता है, जिनका पहले कोई programming का experience नहीं था, special FMS Language के साथ query perform करने के लिए। इनमें से बहुत well known हैं RPG (Report Program Generator) जो Routine Business Reports को Generate करने के लिए बहुत प्रसिद्ध है। RPG को Use करने के लिए User को पहले Required Input fields को define करना पड़ेगा जिसके लिए Input specification format को भरना पड़ता है। इसी प्रकार output fields को specify करने के

लिए output specification formats को भरना पड़ता है। Output को कुछ निश्चित संरचना देने से और default options होने से package याद और use करने के लिए आसान हो जाता है। इस तरह के FMS packages में कुछ प्रसिद्ध उदाहरण है: Mark-4, Data tree, easy tree और power house FMS का structure नीचे दिखाया गया है-



FMS, data management के लिए use होने वाले host operating system के basic access methods पर निर्भर होता है। परन्तु उसकी अपनी एक special language होती है, जो retrievals (डाटा की बहाली) करने के लिए इस्तेमाल की जाती है। यह language कुछ मायनों में standard high level language से ज्यादा powerful होती है। जिस तरीके से यह Data और Development applications को define करती है। इसीलिए, file management system को हम कुछ high level languages से एक level ऊपर मान सकते हैं।

Standard High level languages से तुलना करने पर FMS के कुछ फायदे नीचे दिए गए हैं–

- **Software development पर कम खर्च:** FMS में कम Development cost लगती है High level languages में अनुभवी Programmer भी software system develop करने में महीनों सालों का वक्त लगा देते हैं।
- **कुशल Query facility की support:** Multiple key retrievals (किसी भी Data को पुन: प्राप्त करना), online queries को program करना बहुत tedious (मेहनत) का काम है। यह सही है कि कोई भी व्यक्ति ये भी ध्यान रख सकता है कि FMS की एक कमी यह है कि यह जटिल Mathematical operation और array manipulation को handle नहीं कर पाता। इस कमी को दूर करने के लिए एक Interface देना पड़ता है जो दूसरे programs को call करता है (दूसरे programs की मदद लेता है।) यह program high level language या Assembly language में लिखे होते हैं।

FMS की एक कमी यह भी है कि Data management और access के लिए यह अभी भी basic/पारंपरिक तरीकों तक ही सिमित है। Multiple files में जटिल multiple key queries की सक्षमता के लिए files के बीच जो physical और logical link (सम्बन्ध) चाहिए वह FMS में संभव नहीं है। हालांकि FMS एक सरल, शक्तिशाली

(powerful) tool है, यह high level languages का स्थान नहीं ले सकता, और न ही यह DBMS की तरह complex information को भी Retrieve कर सकता। इसीलिए एक अच्छे Data base system पर निभर करना जरूरी हो जाता है।

2.3 ENTITY RELATIONSHIP (E-R) DIAGRAM

ERD का परिचय

यह एक entity-relationship model है जो डेटाबेस के conceptual view को define करता है और application के semantic feature (शब्दार्थ की विशेषताओं) को analyse (विश्लेषण) करता है, यह features event independent होते हैं। यह Entity-relationship [E-R] Model data Redundancy (Duplicate data/फालतू के Data) को कम करता है। इसे E-R Diagram भी कहते हैं क्योंकि यह entities को graphical रूप से represent करता है, जैसे entity classes को Rectangle (आयताकार) से, relationships को Diamond से [♦] और attributes को circle (वृत्त) या oval (अंडाकार) से। Complex situations के लिए एक Partial E-R Diagram को भी इस्तेमाल किया जा सकता है, जिसमें entities और relationship की Summary (सार) तो होती है परन्तु attributes की Details नहीं होती।

E-R Diagram किसी भी application में entities के मध्य की relationship को देखने के लिए आसान तरीका है। यह tool एक Information application Description को Formal Database Schema में बदलने के लिए बहुत उपयोगी सिद्ध हुआ है।

E-R model से किसी enterprise की conceptual schema को describe किया जा सकता है और इसके लिए physical database का design कैसा है उससे कोई मतलब नहीं होता। बाद में E-R Diagrams को कई Models में conceptual schema के रूप में ही बदल दिया गया, जहाँ Database को actually implement किया गया था।

नीचे E-R concept में इस्तेमाल की जाने वाली कुछ Basic terms के बारे में बताया गया है–

- **Entity:** एक entity कोई भी वस्तु, आदमी या स्थान हो सकता है जो वास्तव में मौजूद है और दूसरे से अलग है–कोई भी object, वातावरण में मौजूद कोई वस्तु (something in the environment).
- **Entity Instance:** Instance का मतबलब है कि एक खास entity का होना (occurance) जैसे कि प्रत्येक व्यक्ति एक entity का Intance है या एक कार का Groups है तो प्रत्येक कार उस entity का Instance है।
- **Entity Set:** एक समान entities के एक Group को entity class या entity set कहते हैं। एक entity class में समान विशेषताएँ होती हैं।
- **Attributes:** Attributes entities की विशेषताओं और उनके सम्बन्धों के बारे में बताते हैं।
- **Simple (Scalars):** यह Data की सबसे छोटी Semantic (अर्थपूर्ण) इकाई है। यह singular होती है, जैसे– city।
- **Composite:** यह attributes का Group है जैसे address [street, city, state, zip]
- **Single value:** अगर किसी attribute में एक बार में एक ही value हो सकती है तो उसे single value कहते हैं जैसे, Max-marks की value एक ही हो सकती हे।
- **Multi valve (list):** यह वह चीज है जिसकी बहुत सारी values हो सकती है जैसे– किसी व्यक्ति की [illegible] (skills) बहुत सारी हो सकती हैं।

- **Domain:** एक attribute के लिए permit की जाने वाली सभी unique values के सेट को हम Domain कहते हैं।
  - ➢ एक तरह की possible values का समूह। उदाहरण के लिए अगर Gender कोई attribute है तो उसके लिए Domain [Male, Female, Others] हो सकते हैं क्योंकि ये तीनों ही Possible entries हैं।
- **Relationships:** यह entity classes के बीच का सम्बन्ध है जैसे–"person" और "automobile" के बीच का सम्बन्ध है 'OWNS' मतलब Person owns automobile.
- **Degree:** एक सम्बन्ध (Relationship) की Degree का मतलब है कि उसमें कितनी entities शामिल हैं।
- **Cardinality:** यह बताती है कि Entity class E1 में कितनी चीजे (Instance) Entity class E2 से सम्बन्धित हैं या होनी चाहिए।
- **One-one Relationship:** जब किसी class की एक entity दूसरी class की सिर्फ एक ही entity सम्बन्धित हो तो यह सम्बन्ध one-one relationship कहलाता है। उदाहरण के लिए एक पति के लिए एक ही वैद्य (legel) पत्नी हो सकती है। और एक पत्नी के लिए एक वैध (legel) पति हो सकता है अत: यह one-one सम्बन्ध है।

A One-to-One Association:

जैसे विद्यार्थी के Id No. का विद्यार्थियों से सम्बन्ध

विद्यार्थी का नाम	विद्यार्थी का ID No.
आकाश वाजपेयी	524.77.9870
मनोज कुमार	543.87.4309
अजय कुमार	553.67.3965

यह one-to-one सम्बन्ध है क्योंकि दो विद्यार्थियों का एक नं० नहीं हो सकता।

Many-One Relationships

इस तरह के सम्बन्ध में Class E2 की कोई entity class E1 की एक या उससे ज्यादा entity से सम्बन्धित हो सकती है या फिर किसी भी entity से संबंधित नहीं होती। (परंतु E1 की कोई entity E2 की एक ही Entity से संबंधित होती है।) उदाहरण के लिए एक माँ के बहुत से बच्चे हो सकते हैं परन्तु प्रत्येक बच्चे की केवल एक ही माँ हो सकती है।

Many to Many Relationship:

इस तरह के संबंध में किसी भी class की कोई entity दूसरी Class की कितनी भी entities के साथ संबंधित हो सकती है। जैसे कक्षा और विद्यार्थियों का उदाहरण। कोई विद्यार्थी कई प्रकार की कक्षाएं ले सकता है, और किसी एक कक्षा में कई प्रकार के विद्यार्थी हो सकते हैं।

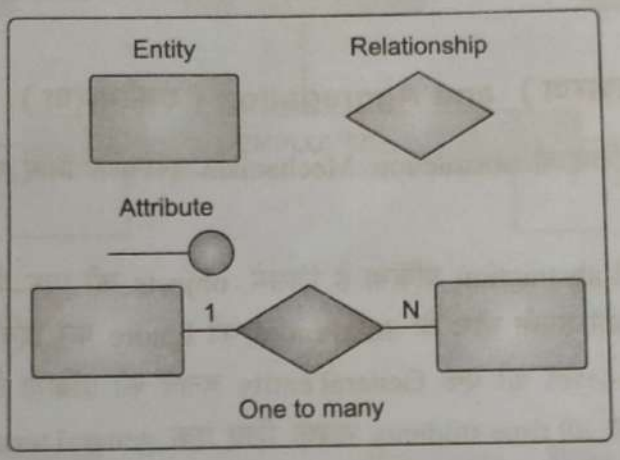
Isa Hierarchies

यह एक प्रकार का विशेष संबंध है जो inheritance attribute (वह गुण या property जो विरासत में मिलती है) प्रदान करता है। जैसे, Truck is a automobile. यहाँ is a Truck or automobile के बीच का संबंध है। अगर automobile में serial no. model आदि होने जरूरी हैं तो मतलब ये सब truck में भी होना जरूरी है। यह चीजें truck को automobile होने के कारण विरासत में मिली है।

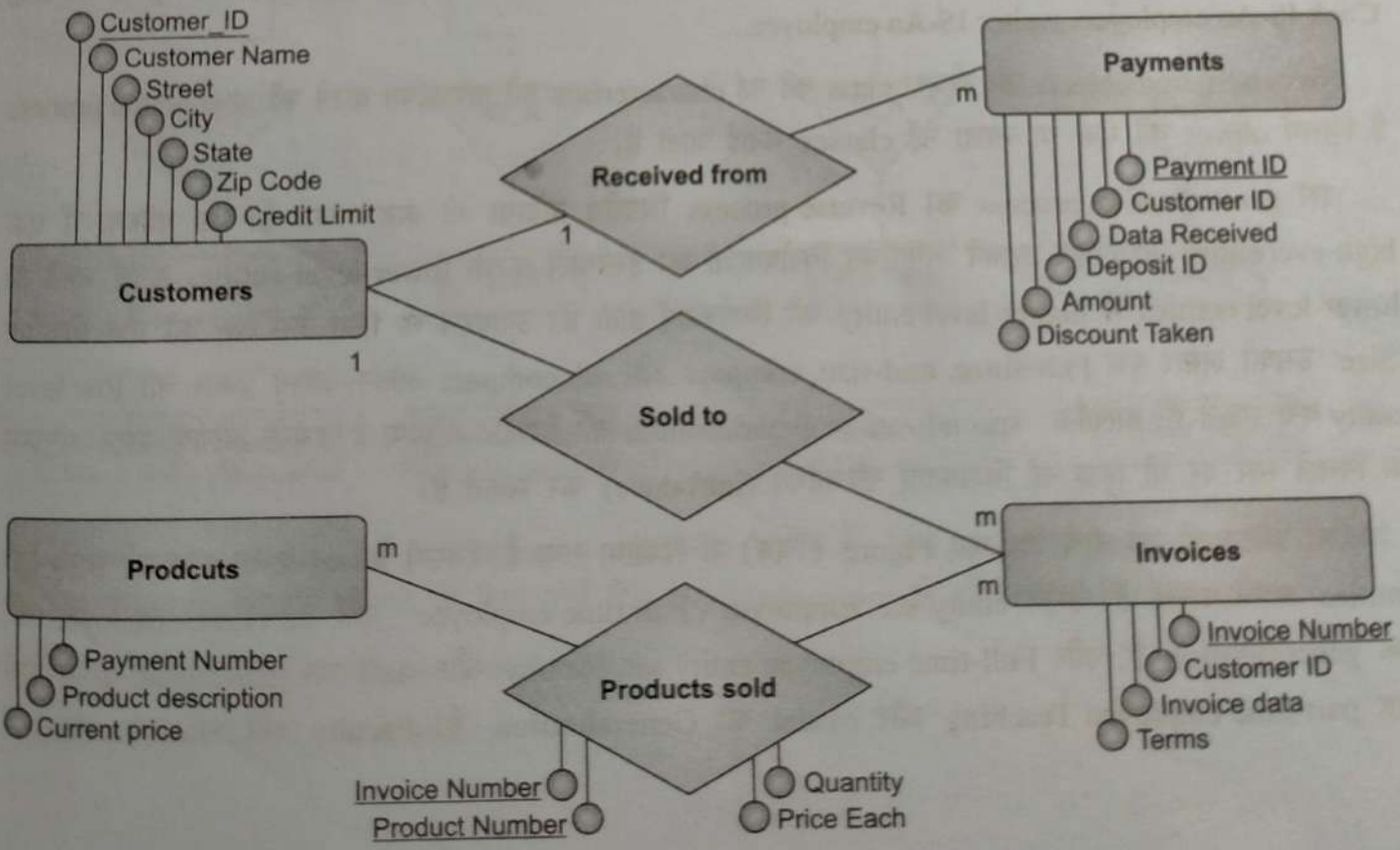
Entity-Relationship Diagram

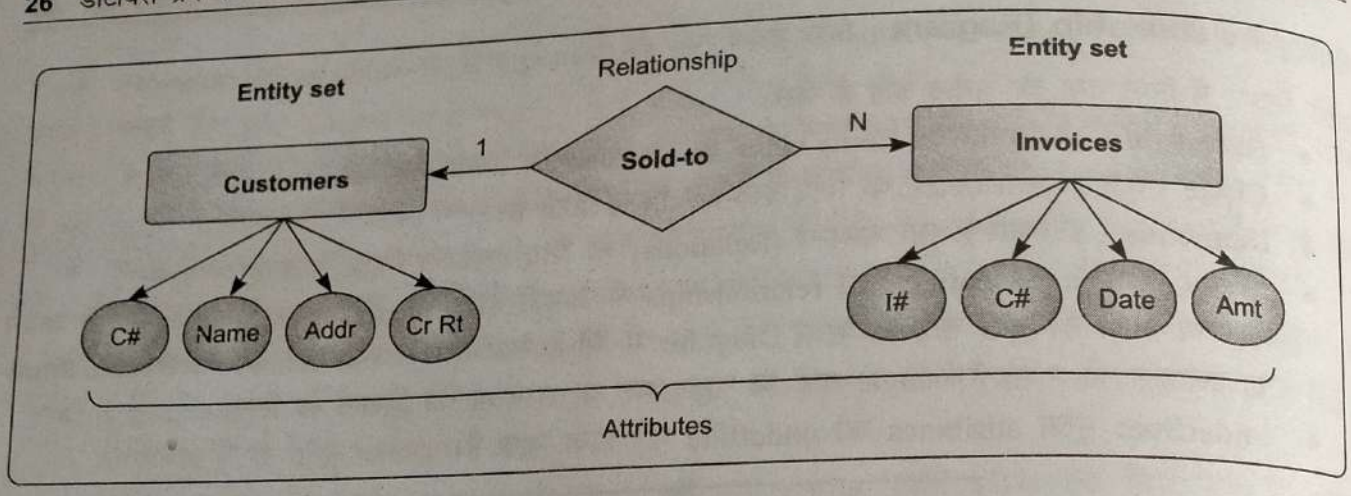
इसके चित्रण में निम्न तरह के संकेत होते हैं जैसे:

- **Rectangle (आयताकार):** Entity class के लिए इस्तेमाल किया जाता है।
- **Circle (वृत्त):** Attributes के लिए इस्तेमाल किया जाता है।
- **Diamonds (ईंटाकार):** को सम्बन्ध (Relations) के लिए इस्तेमाल किया जाता है।
- **Arcs (रेखाएँ):** ये entities को relationships से जोड़ती हैं। ये attributes को entities से भी जोड़ने के काम आती हैं। कुछ तरह के E-R Diagram में सीधी रेखाओं की जगह arrows या Double-arrow का इस्तेमाल भी करते हैं। single एक या बहुत तरह के सम्बन्धों को दिखाने के लिए।
- **Underline:** मुख्य attributes को underline कर दिया जाता है।



E-R Diagram का उदाहरण





Generalization (सामान्यकरण) and Aggregation (एकीकरण)

Model की Information के लिए दो abstraction Mechanism इस्तेमाल किए जाते हैं– Generalization और Aggregation.

Generalization एक ऐसी abstraction प्रकिया है जिसमें, objects की एक जैसी (general) characteristics (विशेषताओं) को देखा जाता है और उनके बीच के differences को Ignore कर दिया जाता है। यह lower-level की entities के एक समूह से High-level की एक General entity बनाने की प्रक्रिया है। उदाहरण के लिए, graduate या undergraduate, full time या all time students सबके लिए एक general term 'student' का इस्तेमाल किया जा सकता है। इसी प्रकार objects जैसे Cook, waiter, cashier इत्यादि का generalisation किया जाए तो इन्हें सभी के लिए 'Employee' का इस्तेमाल किया जा सकता है। Generalisation एक IS-A Relationship है इसलिए, Cook IS-An employee, waiter IS-An employee....

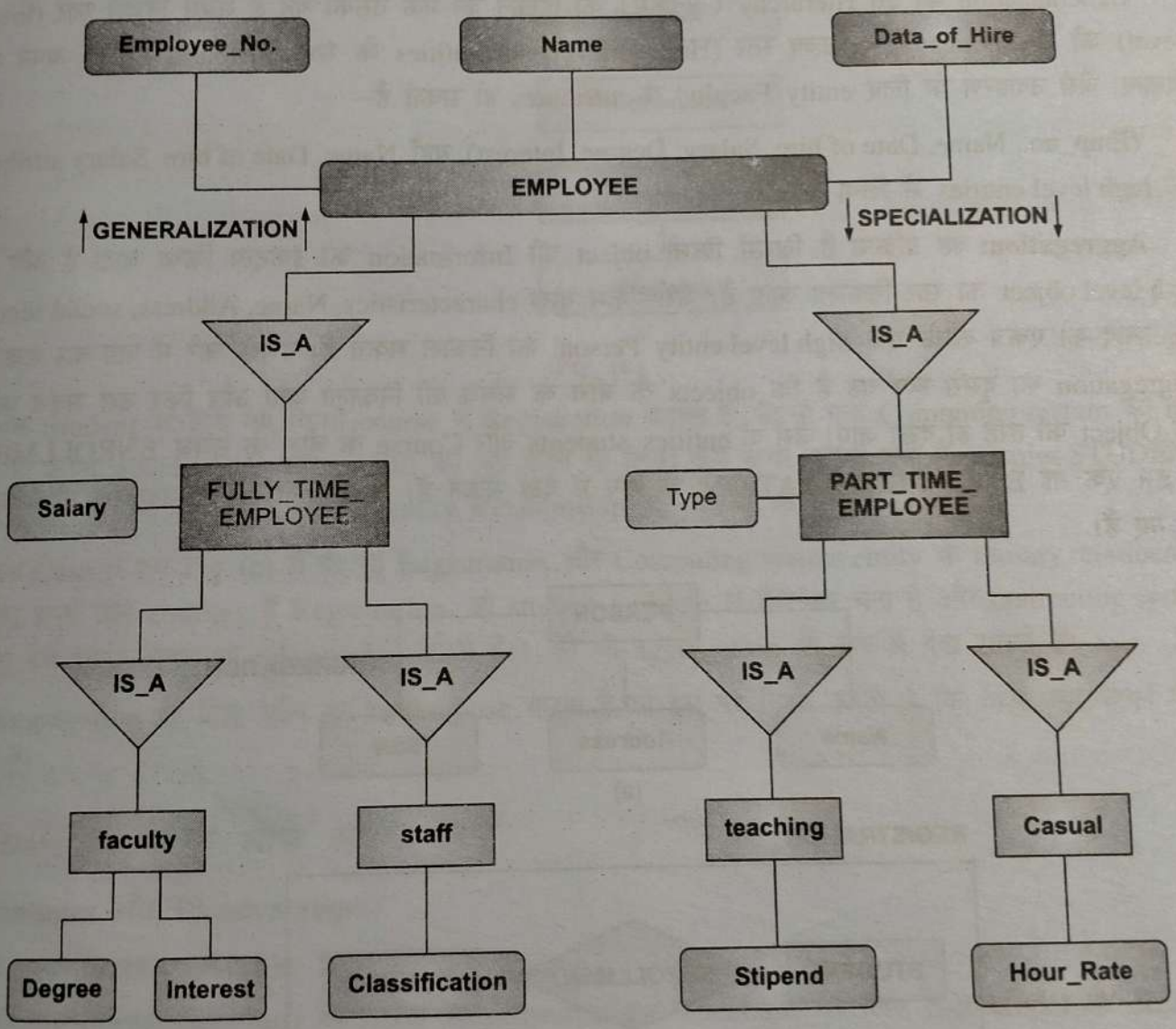
Specialisation objects की मौजूदा class की नई characteriscs को परिभाषित करने की abstraction process है जिसमें object की एक या ज्यादा नई classes बनाई जाती है।

इसे generalisation process का Reverse process विपरीत प्रक्रिया भी कह सकते हैं। इस प्रक्रिया में एक high-evel entity को लेकर उसकी अतिरिक्त विशेषताओं का इस्तेमाल करके lower-level-entities बनाई जाती है। lower-level entities में higher-level entity की विशेषताएँ होती हैं। उदाहरण के लिए जैसे car की एक विशेषता 'Size' उसको लेकर हम Full-siuze, mid-size, compact और sub compact अलग-अलग प्रकार की low level entity बना सकते हैं। हालांकि specialization, generalisation की विपरीत प्रक्रिया है, इसके अलावा इसमें अनुक्रम के निचले स्तर पर भी कुछ नई विशेषताएँ परिभाषित (Introduce) कर सकते हैं।

दोनों प्रक्रियाओं को नीचे दिए गए Figure (चित्र) में दिखाया गया है, जिसमें lower-level (निचले स्तर) की entities अलग-अलग हैं। इसमें entity set 'Employee', Full time employee और part time employee का एक generalisation है, और Full-time employee entity set, Faculty और staff का generalisation है इसी तरह part-time employee Teaching और casual का Generalisation है। Faculty और Staff Entities में

entity set Full time employee की विशेषता (attribute) जैसे Salary समाहित है मतलब Salary इन दोनों का भी Attribute है और इसी तरह entity Full-time employee में Entity Employee की सारी विशेषताएँ (attributes) होती हैं। अब क्योंकि 'Employee', Full-time employee का generalisation है इसका विपरीत करें तो, Full-time employee, entity class 'Employee' का specialisation है, जिसकी अपनी एक अलग विशेषता (attribute) है। 'Salary' इसी प्रकार Part-time employee भी 'employee' का एक specialisation विशेष प्रकार है। जिसका अलग से attribute है 'Type'।

Database के Design करते वक्त यह जरूरी है कि Data modelling Schema Generalisation को व्यक्त (दिखा) कर सके।



Generalisation entities की एक hierachy (अनुक्रम) बनाता है, जिसे table की hierarchy के द्वारा भी दर्शाया जा सकता है। जिन्होंने सुविधा के लिए नीचे दिए गए relationships से भी दिखाया जा सकता है।

EMPLOYEE (Empl-no, name, Date-of-hire)

FULL-TIME (Empl_no, salary)

PART-TIME (Empl-no, type)

FACULTY (Empl-no, Degree, Interest)

STAFF (Empl-no, Hour-rate)क

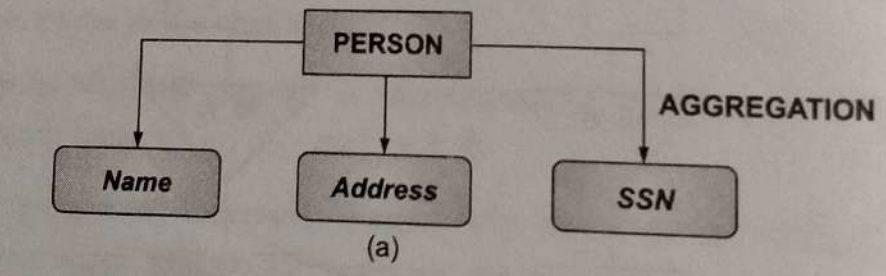
TEACHING (Empl-no, Stipend)

यहाँ प्रत्येक entity की Primary key एक अलग table से मेल खाती है और उस table से मेल खाती है और उस table की एक विशेष Row को Indicate करती है।

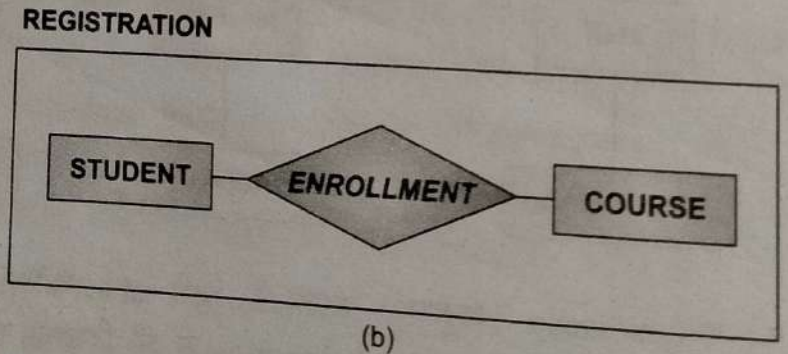
Generalisation की इस Hierarchy (अनुक्रम) को दिखाने का एक तरीका यह है सबसे निचले स्तर (lowest level) की जो entities है, वो उच्च स्तर (Highest level) की entities के सभी attributes (गुण) अपने अंदर दिखाए। जैसे उदाहरण के लिए entity 'Faculty' के attributes हो सकते हैं–

(Emp_no., Name, Date of hire, Salary, Degree, Interest) यहाँ Name, Date of hire Salary attributes को high level entities से लिया गया है।

Aggregation: वह प्रक्रिया है जिसमें किसी object की Information को इकट्ठा किया जाता है और फिर high level object का सार निकाला जाता है। जैसे– हम कुछ characteristics, Name, Address, social security no. आदि को एकत्र करके एक high level entity 'Person' को निकाल सकते हैं। उसके बारे में पता कर सकते हैं। Aggregation का दूसरा रूप यह है कि objects के बीच के संबंध को निकाला जाए और फिर उस संबंध को भी एक Object की तरह ही देखा जाए। जैसे दो entities: students और Course के बीच के संबंध 'ENROLLMENT' को हम एक नई Entity 'REGISTRATION' के रूप में देख सकते हैं। Aggregation के उदाहरण नीचे चित्र में दिए गए हैं।



(a)



(b)

अब नीचे दिए गए चित्र की Relationship computing को देखिए। यहाँ हमारे पास Entities, STUDENT, COURSE, और COMPUTING SYSTEM के बीच एक संबंध (Relationship) है।

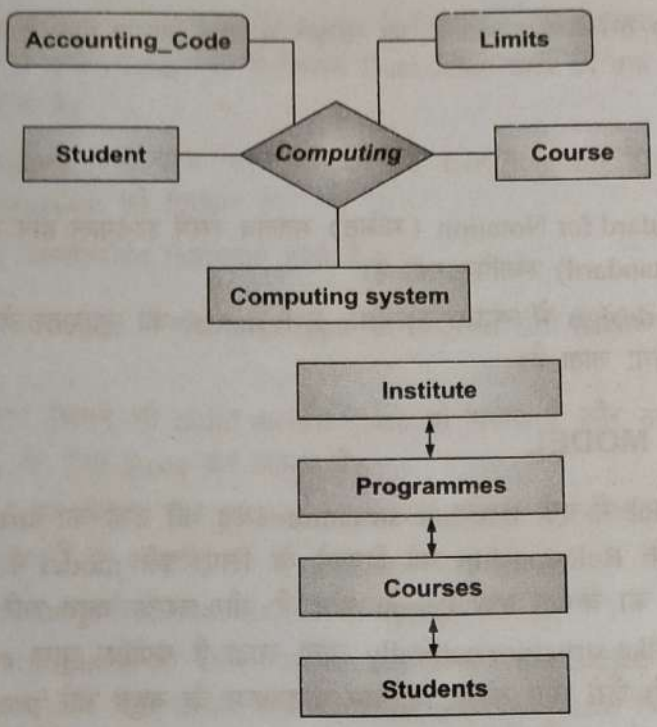


Fig. (c)

एक Student जो दिए गए किसी course में Registration कराता है, किसी एक Computing system को use करता है, अपने assignments और project को पूरा करने के लिए। जैसे ऊपर बताया गया है, entities STUDENT और COURSE को aggregate करके entity 'Relationship' को बनाया जा सकता है।

इस Case में हम Fig. (c) में दी गई Registration और Computing system entity के ternary relationship (त्रिगुट: इसमें तीन entitites हैं Registration, जो student + course से मिलकर बना है और computing system इसलिए इसे त्रिगुट संबंध या relationship कहते हैं।) को भी aggregation के रूप में देख सकते हैं।

Aggregation के लिए कौन सा Method use करना है वो इस पर निर्भर करता है कि आप क्या व्यक्त करना चाहते हैं।

E-R Data Model के लाभ और हानियाँ

(Advantages और Disadvantages)

ER Model के लाभ (फायदे):

- **संबंधों (relationship) का सरल Representation:** Database के लिए E-R Model के design में Database Model के relationships का representation (प्रदर्शन) बहुत सरल तरीके से किया गया है।
- **E-R से दूसरे किसी Data Model में आसानी से रूपांतरण (बदलाव):** ER Diagram को Network या Hierarchial Data Model में आसानी से बदला जा सकता है।
- **अच्छी तरह समझने के लिए Graphical representation:** एक E-R model विभिन्न Entity उनके Attribute और entitites के बीच के Relationships को Graphical (चित्रात्मक) और Diagrammatical

(आरेख के रूप में) तरीके से दर्शाता है, जो समझने में बहुत आसान होता है और इससे गलतियाँ बहुत कम होती हैं।

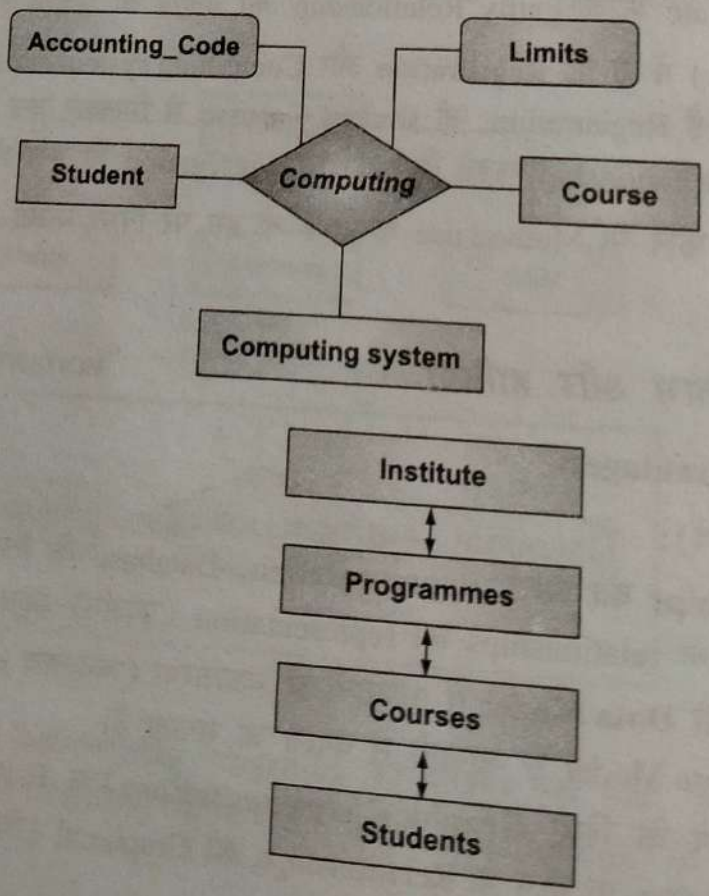
E-R Model से हानियाँ

E-R से हानियाँ हैं–

- No Inudsutry standard for Notation (संकेत) मतलब इसमें इस्तेमाल होने वाले संकेतों (Notations) के लिए कोई मानक (standard) स्थापित नहीं है।
- **केवल High-level design में ज्यादा उपयोग:** E-R model को ज्यादातर केवल High level design के लिए ही इस्तेमाल किया जाता है।

2.4 HIERARCHICAL MODEL

Hierarchical data model में एक tree-like-strucutre (पेड़ की तरह का structure) का इस्तेमाल किया जाता है, records के बीच के Relationship को दिखाने के लिए। इसे model में Parent_child relationship होती है। इसमें प्रत्येक entity का केवल एक Parent होता है और उसके बहुत सारे Children होते हैं। बहुत से Data organisation में tree-like structure naturally पाया जाता है क्योंकि कुछ entities स्वाभाविक रूप से ही Hirarchical order में होती हैं। जैसे एक संस्था के पास पाठ्यक्रम के बहुत सारे programmes हैं। हर program में कुछ courses हैं। प्रत्येक course में कुछ students ने registration किया है। नीचे दिए गए figure में चारों तरह की entities Instritue, Program, course और student का Hierarichal structure दिखाया गया है। इसमें यह दर्शाया गया है कि किसी Institute का database कैसे दिखाई देता है मतलब उसका Database occurance दिखाया गया है। Database occurance से मिलकर ही Database बनता है।



Hierarchical database एक records का collection है जिसमें Records एक दूसरे से एक link के द्वारा जुड़े होते हैं। प्रत्येक Record में कुछ Fields होते हैं जिसमें Data value होती है। एक Link दो Records के बीच का association (जुड़ाव) होता है।

एक tree structure diagram वही काम करता है जो एक E-R diagram करता है, यह नाम के अनुरूप Database के पूरे logical structure को दिखाता है।

Hierarchical model में निम्नलिखित विशेषताएँ होती हैं–

- प्रत्येक Hierarchical structure में एक Root type का Record होता है और इस Record का कोई Parent Record नहीं होता।
- Root record के लिए कितने भी child record types हो सकते हैं और उनमें से प्रत्येक एक छोटे tree structure (subtree) के लिए Root बन सकता है।
- प्रत्येक Child record का केवल एक Parent record हो सकता है। इसका मतलब दो records के बीच M : N Relationship नहीं हो सकती।
- Parent record का Data सभी Child records पर apply होता है।
- एक Child record occurance के लिए Parent record की occurance होना जरूरी है, और parent record को delete करने के लिए उसके सभी child record occurance को भी Delete करना जरूरी है।

Hierarchical Model के फायदे

- यह समझने में बहुत सरल होता है।
- Relational data model से Hierarchical model का प्रदर्शन (performance) बेहतर है।

Hierarchical Model की कमियाँ (Disadvantages)

- निचले स्तर की Value तक पहुँचना बहुत मुश्किल होता है।
- Organisation की बदलती हुई जरूरतों को पूरा करने के लिए यह Model Flexible नहीं है।
- अगर Parent node (records) को delete करना है तो child node को भी जबरन Delete करना पड़ेगा।
- Pointers की storage के लिए extra space चाहिए।

2.5 NETWORK MODEL

Network model को एक संस्था Database Task Group of the Conference on Data System Language (DBTG/CODASYL) ने 1960 में Formalize किया था। उनकी पहली report कई revise हुई, जिसमे network data model की विस्तृत detailed specifications दी हुई थी। इन specifications को पूरा करने वाले Model को DBTG data model भी कहते है। इस report और उसके बाद के revisions में जो specifications थी उन पर बहुत बहस और आलोचनाएँ हुई। बहुत सारी current database applications commercial DBMS systems पर DBTG model का इस्तेमाल करके बनाई गई हैं।

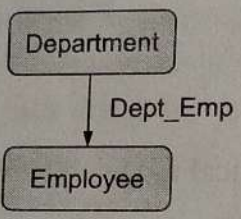
DBTG Set: DBTG model database की entities और उनके संबंधी (relations) को दिखाने के लिए दो अलग तरह के data structures का इस्तेमाल करता है: record type का और set type का। Record type यह बताताहै कि

entity किस प्रकार की है, यह बहुत से data items से बना होता है जो entities के attributes (विशेषताओं) को दर्शाते है।

Set का इस्तेमाल, दो record types जिनमें एक owner record type (मालिकाना record) और दूसरा member record type (सदस्य record) होता है, उनके बीच के सीधे संबंध को दर्शाने के लिए किया जाता है। Set type में भी record type की तरह, जैसा उसके नाम से पता चलता है, One-to-many relationship (एक का संबंध अनेक से) (1: M) होती है। यह relationship owner और member record type के बीच होती है। एक set type के member के रूप में बहुत record हो सकते है पर केवल एक record type ही दिए गए set का owner हो सकता है।

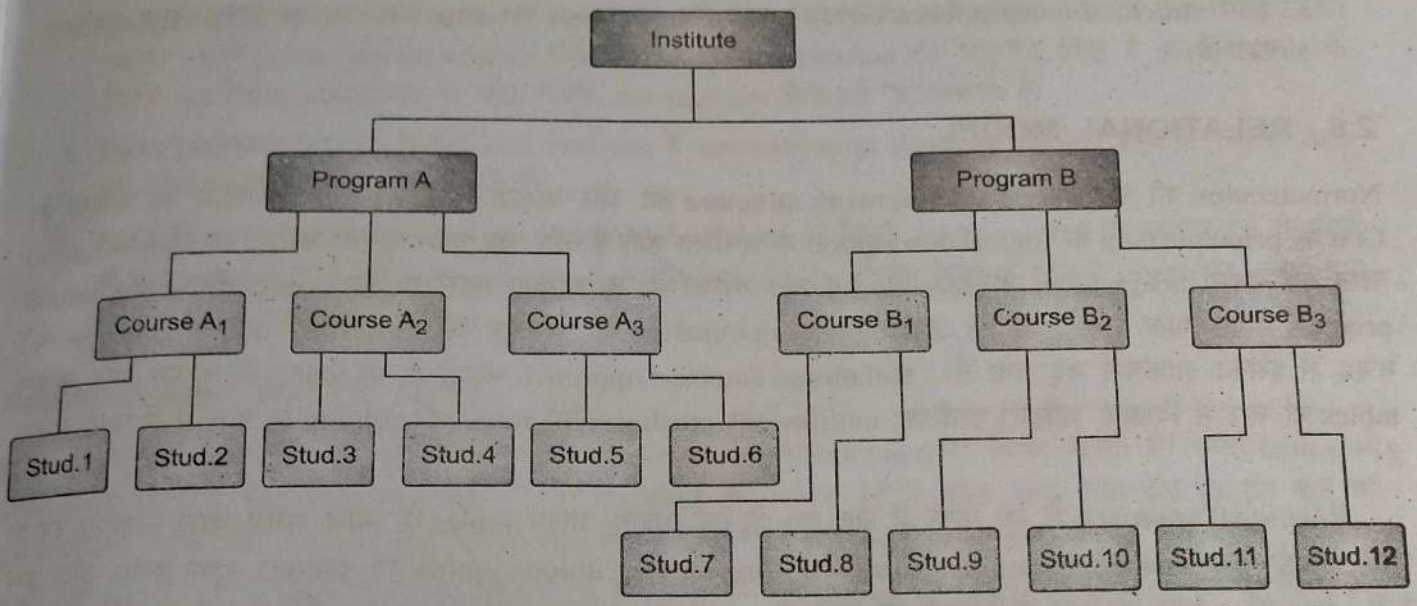
एक database में एक या उससे ज्यादा record types और set type हो सकते हैं। एक set types का मतलब एक owner record type और उसके member record types की कितनी बार भी occurrence (पाया जाना)। एक record type एक ही set type में दो जगह नहीं पाया जा सकता।

Bachman ने एक data structure, diagram graphics की मदद से Introduce किया था, set के द्वारा बनाई गई logical relationships को denote करने के लिए। यहाँ पर एक labeled rectangle (आयताकार) entity या record type को दर्शाता है। एक arrow (तीर का निशान) जो दो rectangles को जोड़ता है, set type को दर्शाता है। Arrow की दिशा owner record type से Member record type की तरफ होगी। नीचे दिया हुआ Fig दो record types दिखा रहा है, (DEPARTMENT और EMPLOYEE) और एक set type DEPT-EMP दिखा रहा है जिसका owner record type department है और member record EMPLOYEE है।



Data structure diagrams को record type rectangle में और field names शामिल करके बढ़ाया जा सकता है, और arrow को set में शामिल data fields को Identify करने के लिए इस्तेमाल किया जाता है। एक one-to-many (1: M) relationship को set arrow द्वारा दिखाया जाता है। जो owner record type के owner field से शुरू होता है, Arrow member record type के member field पर खत्म होता है। वह field जो relationship को support करता है उसका पता स्पष्ट रूप से लग जाता है। एक Logical record उसी नाम से E-R diagram में प्रत्येक entity को दर्शाता है। Record के Data field entity के attributes को दर्शाते हैं। हम term logical records का इस्तेमाल यह दिखाने के लिए करते है कि वास्तिवक implementation बिलकुल अलग हो सकता है।

किसी E-R diagram को network diagram में बदलने के लिए प्रत्येक me : M binary relationship को एक set में बदलना है। (एक 1 : 1 binary relationship, एक 1 : M relationship का विशेष case है) अगर Entity type E1 से Entity type E2 में एक 1 : M binary relationship R1 है तो binary relationship को कई घटनाओं (instances) के Set के रूप में दर्शाया जा सकता है, जिसका एक Instance S1 ENTITY E1 के अनुरूप होता है। (E1 यहाँ owner record की तरह काम करती है), और बहुत सारे Instances Entity E2 के अनुरूप होते है, जो Member की तरह काम करते है। अगर relationship के attributes है, तो जब तक attributes member record types को नहीं सौंपे (assign) जाते, तब तक उन्हें एक अलग logical record type में पड़ता है। इसके अतिरिक्त record type का परिचय कराने के लिए, original set को दो एक जैसे sets में बदलना पड़ेगा, जिनमें relationships के attributes के अनुरूप कुछ records है जो member की तरह काम करते है, दोनों sets में, और entities के अनुरूप कुछ records है, जो owner की तरह काम करते है।



Network Model के फायदें:

Network model के मुख्य फायदे है;

1. **Conceptual simplicity (concept की सरलता):** Hierarchical model की तरह ही, network model भी conceptually बहुत सरल और आसान है design करने के लिए।
2. **और अधिक relation type को संभालने की क्षमता:** Network model one to many (एक से अधिक) और many to many (बहुत से बहुत) relation को handle कर सकता है, जो real life situations को Model (नमूना बनाने) करने के लिए बहुत फायदेमंद है।
3. **आसान Data access:** इसका data access (data तक पहुँच) hierarchical model की अपेक्षा सरल और लचीला है।
4. **Data integrity (Data की अखंडता):** Network बिना owner के किसी member को मौजूद रहने की अनुमति नहीं देता।
5. **Data की independence:** Network model programs को जटिल Physical storage details से अलग करने में hierarchical model से बेहतर है।
6. **Database standards (मानक):** यह एक standard model हैं।

Network Model की हानियाँ:

1. **System की जटिलता:** सारे records pointer का इस्तेमाल करके संभाले जाते है, इसलिए पूरे database का structure बहुत जटिल हो जाता है।
2. **Operational Anomalies:** (काम करने की विसंगतिया/दिक्कते); किसी record को Insert, delete या update करने के लिए बहुत सारी pointer की adjustment करनी पड़ती है।

3. इसमें structural independence (स्वतंत्र) नहीं है– database का structure change करना बहुत मुश्किल होता है।

2.6 RELATIONAL MODEL

Normalization की तकनीक के द्वारा network structure को और बढ़िया (साफ-सुथरे) में बदला जा सकता है। Cod के principle data के logical description से संबंधित होते है और यह ध्यान रखना चाहिए यह data को store करने का सबसे independent और feasible (संभव) तरीका है। कुछ साल पहले ही इस concept का commercial projects में इस्तेमाल शुरू हुआ है। इसकी data organisation की सरलता और प्रभावशाली query language की वजह से इसका आकषर्ण बढ़ गया है। Relational database approach कहती है कि data को केवल और केवल tables के रूप में दिखाना चाहिए। इसलिए, entities और attributes को rows और column के रूप में दिखाना पड़ता है।

Relational approach में जो फर्क है वह यह है कि अलग-अलग tables के बीच संबंध कैसे स्थापित किया जाए। इससे कुछ mathematical operations जैसे, projections, unions, joins का इस्तेमाल शुरू हुआ। और इस तरह data को tables के रूप में दिखाने के लिए एक technique normalisations का इस्तेमाल किया गया।

Normalisation technique data items जैसे records, segments और tulles को समूह में बाटंने के लिए इस्तेमाल की जाती है, यह जरूरी है क्योंकि relational model में data items table में व्यवस्थित होते है, जो structure बनाते हैं।

Relationship और integrity नीचे दिए गए तरीके से defined होते है:

1. किसी भी एक column में सभी item एक तरह के होने चाहिए।
2. प्रत्येक item या तो number (संख्या) होनी चाहिए या कोई character string.
3. Table की प्रत्येक row अलग होनी चाहिए।
4. Table की प्रत्येक row के क्रम का कोई महत्व नहीं होना चाहिए।
5. Table के columns का नाम अलग-अलग होना चाहिए और उनके क्रम से भी कोई फर्क नहीं पड़ना चाहिए।
6. अगर table में N columns है, तो उसकी Degree N होती है। जिसे कभी-कभी cardinality भी कहा जाता है। कुछ base tables से हम relation set कर सकते है, views बना सकते है, जो एक ही database के अलग-अलग users को जरूरी information प्रदान करते हैं।

Relational approach के लाभ:

इसकी लोकप्रियता इसके बहुत प्रकार के products के कारण है। इसके कुछ फायदे है:

1. **Ease of use:** इस्तेमाल करने में आसान यह rows और columns के कारण इस्तेमाल करने में बहुत आसान होता है।
2. **Flexibility:** (लचीलापन) अलग-अलग tables, जिनमें information link करनी या निकालनी होती है, सरलता से बदली जा सकती है project और join जैसे operators का इस्तेमाल करके।
3. **Precision:** Relational algebra और calculus का इस्तेमाल होने से data बिलकुल Precise (शुद्ध) होता है, इसमें गलतियाँ बहुत कम होती है।

4. **Security (सुरक्षा):** अलग tables में sensitive data को store करके उसे सुरक्षित रखा जा सकता है। उसका अपना अलग authorization control होता है। अगर authorization की अनुमति होती है तो इस table के किसी attribute (coloumn) से link करके information निकाली जा सकती है।
5. **Data Independence:** Relational database में normalisation structure इस्तेमाल होता है जिससे data की स्वतंत्रता हासिल की जा सकती है।
6. **Data Manipulation Language:** Relational database में किसी ad-hoc query का जत्राब देने के लिए जो language होती है वह relational algebra और calculus पर निर्भर करती है जिससे query का जवाब देना आसान हो जाता है।

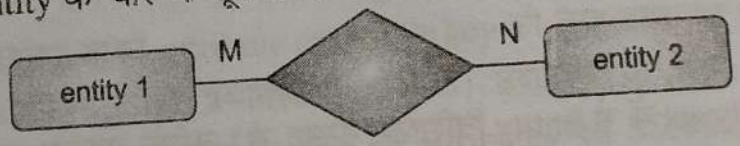
Relational Approach की हानियाँ:

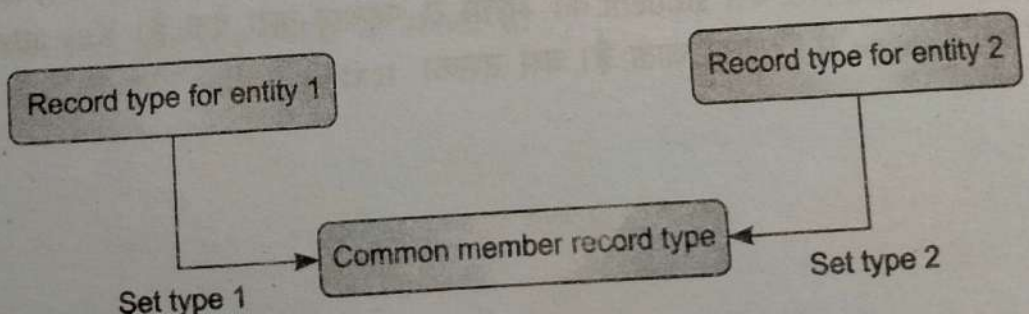
Relational approach का एक नुकसान यह है कि machine performance पर निर्भर करता है। अगर tables की संख्या बहुत ज्यादा है या tables बहुत बड़ी है तो यह query को जवाब देने में बहुत वक्त लगा देता है। इस बात की तारीफ करनी चाहिए कि relational data approach की सरलता उसके logical view में हैं। किसी भी Interactive system के साथ, जैसे join की तरह का operation यह physical storage पर भी निर्भर कर सकता है। इसलिए relational database में यह आम है कि, databases को इस तरह tune किया जाए कि physical data layout को अच्छा प्रदर्शन करने के हिसाब से चुना जाए। हालांकि Relational database approach logically आकर्षित करती है और commercially भी feasible है, परन्तु अगर data प्राकृतिक रूप से hierarchical तरीके से organised है और hierarchical model से और बेहतर परिणाम मिल सकते है। आगे के sections में relational or non-relational approach की तुलना इसे समझने में मदद करती है।

Relational Model का एक उदाहरण

नीचे के चित्र में दिए गए उदाहरण से हम RDBMS की कुछ विशुषताओं को देखते है। किसी relation की निम्नलिखित विशेषताएँ होती है:

1. एक table two-dimensional structure होता है जिसमें row and columns होते है।
2. प्रत्येक column का एक अलग नाम होता है। (attribute का नाम) और columns के क्रम का कोई फर्क नहीं पड़ता।
3. प्रत्येक row एक (tulle) entity set की single (केवल एक) data entity को प्रर्दशित करती है।
4. प्रत्येक row और column जहाँ मिलते है वहाँ एक data value होती है।
5. प्रत्येक row में एक entity के बारे में सूचना होती है।





जैसा figure में दिया गया है, tulle एक relation एक row की values का collection है। Tulle एक record instance के बराबर है। एक n-tulle n attribute की values होती है, जहाँ relation की degree कहलाता है। product 4-tuple का एक उदाहरण है। एक relation में कितने tulles है यह उसकी cardinality कहलाता है।

Domain किसी attributes की possible values का set है। उदाहरण के लिए एक relation product में QUANTITY-ON-HAND के लिए domain शून्य या शून्य से बड़ें अंको का set है।

किसी relation VENDOR में CITY attribute की domain alphabetic (वर्णमाला) strings का set है।

PRODUCT (PRODUCT#, DESCRIPTION, PRICE, QUANTITY-ON-HAND)

VENDOR (VENDOR#, VENDOR-NAME, VENDOR-CITY)

इस तरीके में, attributes (या उनका समूह) जिनके लिए प्रत्येक tulle की value अलग-अलग है, वह primary key कहलाते हैं। सरलता के लिए primary key, attributes को underline कर दिया जाता है। Relational data model में किसी primary attributes की value NULL (खाली नहीं हो सकती। जैसे students का Roll no. same नहीं हो सकता ना ही खाली हो सकता है। हालांकि कुछ अलग attributes उन्हें candidate keys कहा जाता है। primary key की तरह काम कर सकते है, एक या एक से ज्यादा के समूह को एक बार में चुन कर इस्तेमाल किया जा सकता है, इन बाकी keys को alternate keys कहा जाता है।

कभी-कभी relation में दो attributes को combine करने के बाद ही tuple (row) की uniqueness identify होती है। मान लिजिए अगर हम सिर्फ एक class के student की बात कर रहे है तो केवल Roll No को primary key कर लिया जा सकता है, परन्तु हमारे tables में बहुत सारी classes शामिल है, तो प्रत्येक class में same roll no. के students होगे ऐसे में class और roll no के combination को primary key की तरह इस्तेमाल किया जा सकता है।

2.7 ATTRIBUTES (COLUMN या ENTITIES के गुण)

Attribute किसी entity की विशेषता (गुण) बताता है। ER Diagram में attributes को oval shape में दिखाया जाता है।

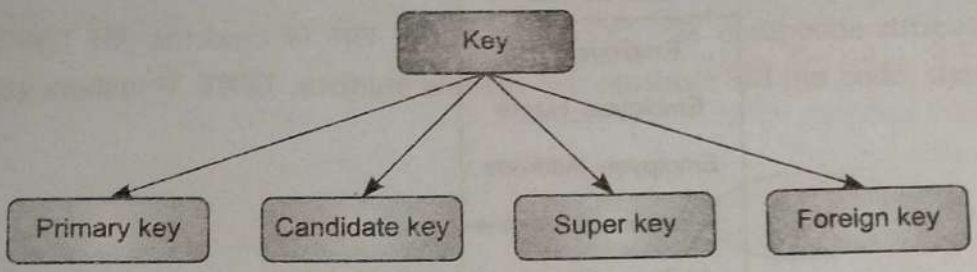
Attributes चार प्रकार के होते है:

1. Key attribute
2. Composite attribute
3. Multivalued attribute
4. Derived attribute

1. Key Attribute

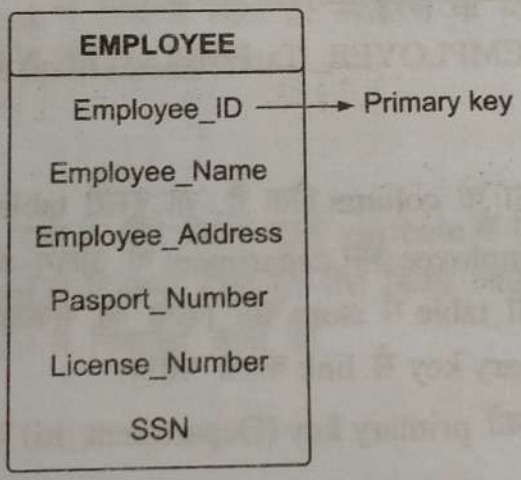
एक key attribute किसी entity set में से entity विशिष्ट (अलग से) तरीके पहचान करा सकता है। जैसे student के एक set में से student का Roll No. उस student की अलग से पहचान करा देता है। Key attribute को भी बाकी attributes की तरह oval shape से दिखाया जाता है। बस उसका text underline होता है।

Key के प्रकार:



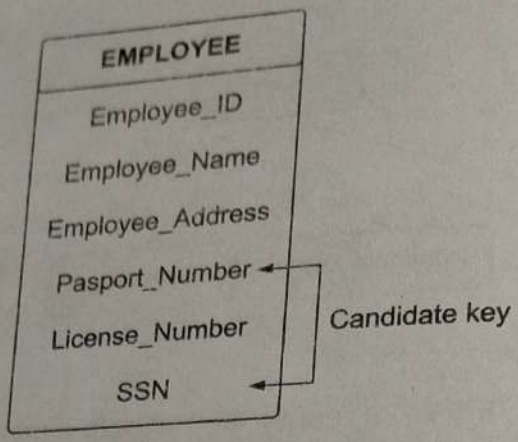
(a) Primary Key

- यह पहली key होती है जो केवल एक entity को विशिष्टता (अलग से) दिखाने के लिए इस्तेमाल की जातीहै, जैसा हम person table में देख सकते है, एक entity के लिए बहुत से keys हो सकती है। इसमें से जो सबसे उपयुक्त होती है वह primary key होती है।
- EMPLOYEE table में ID primary key होती है, क्योंकि यह प्रत्येक employee के लिए अलग होती है। EMPLOYEE table में हम License Number या Passport Number को भी primary key s की तरह इस्तेमाल कर सकते है क्योंकि यह भी अलग होती है।
- प्रत्येक entity के लिए, Primary key का चुनाव जरूरत पर और developer पर निर्भर करता है।



(b) Candidate Key

- Candidate key कोई attribute या attribute का set है, जो किसी tuple (row) को अलग से दिखा सकते है। (identify कर सकते है।)
- Primary key को छोड़कर बाकी attribute candidate key की तरह इस्तेमाल किए जा सकते है। Candidate keys, Primary key की तरह ही प्रभावशाली होती है। उदाहरण के लिए: EMPLOYEE table में, ID primary के लिए सबसे ज्यादा उपयुक्त है। बाकी के attributes जैसे SSN Passport Number और License Number को candidate key बताया जा सकता है।



(c) Super Key

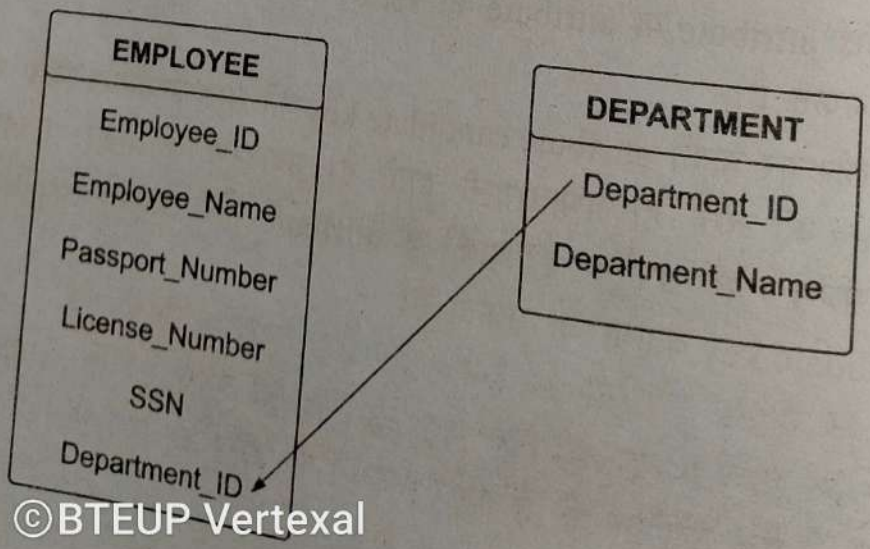
Super key attribute का set है, जो किसी Row (tuple) की अलग से पहचान करा सकता है। यह candidate key का super set होता है।

उदाहरण के लिए: ऊपर दी गई EMPLOYEE table में, अगर EMPLOYEE_ID और EMPLOYEE_NAME की बात की जाए तो, दो employees का नाम एक जैसा हो सकता है, परन्तु EMPLOYEE_ID कभी भी एक नहीं हो सकती, तो इन दोनों का combination भी एक key हो सकती है।

Super key EMPLOYEE-ID, (EMPLOYEE_ID, EMPLOYEE-NAME) आदि हो सकती है।

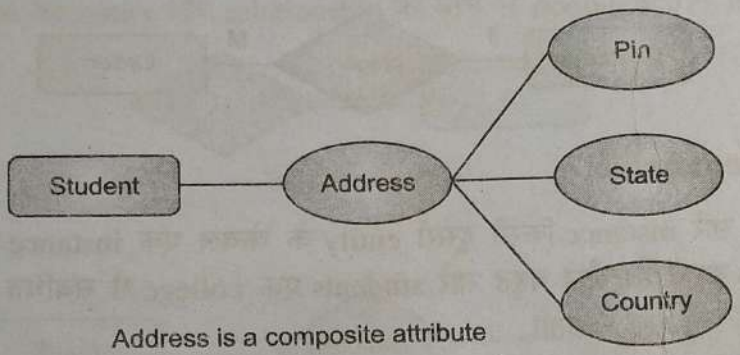
(d) Foreign Key

- Foreign keys किसी table का वो column होता है, जो दूसरी table में primary key होता है।
- किसी company में प्रत्येक employee और department दो अलग-अलग table होगी क्योंकि department की details को employee की table में store नहीं किया जा सकता। इसलिए हमें दोनों tables को आपस में किसी एक table की primary key से link करना पड़ेगा।
- हम DEPARTMENT table को primary key (Department_Id) को employee table में एक attribute की तरह add कर देते हैं।
- इसलिए EMPLOYEE table में Department_Id एक foreign key होती है और इससे दोनों tables आपस में related होती है।



2. Composite Attribute

यह attribute जो बहुत सारे attribute का मेल (combination) हो, उसे composite attribute कहते हैं। उदाहरण के लिए एक entity student में उसका attribute address कई attribute जैसे pin code, state, country आदि का combination है।



Address is a composite attribute

3. Multivalued Attribute

एक attribute जो अपने अंदर अनेक value को समाहित (hold) कर सकता है, उसे multivalued attribute कहते हैं। इसे double ovals से ER Diagram में दिखाया जाता है। उदाहरण के लिए किसी व्यक्ति के पास एक से ज्यादा mobile number है तो यह attribute multivalued होगा।

4. Derived Attribute

वह attribute जिनकी value बदलती रहती है और किसी दूसरे attribute से निकलती है उन्हें derived attribute कहते है। इन्हे E-R Diagram में dashed oval से दिखाया जाता है। जैसे किसी व्यकित की age समय के साथ बदलती रहती है और दूसरे attribute (Date of birth) से निकाली जाती है।

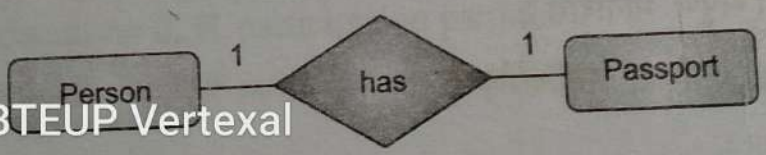
Mapping Constraints

ER diagram में relationship को diamond shape से दिखाया जाता है। यह entities के बीच संबंधों को दिखाता है। Relationships चार प्रकार की होती है।

1. One to One
2. One to Many
3. Many to One
4. Many to Many

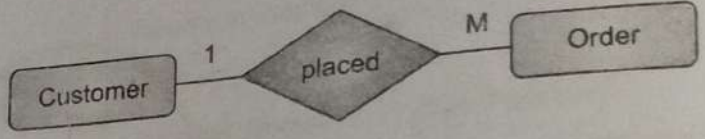
1. One to One Relationship

अगर किसी entity का एक instance (एक उदाहरण) किसी दूसरी entity के केवल एक instance से ही संबंधित होता है, तो उसे one to one relationship कहा जाता है। जैसे एक व्यक्ति का एक ही passport हो सकता है।



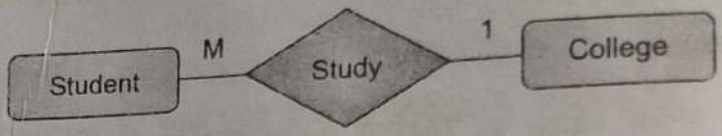
2. One to Many Relationship

अगर किसी entity का एक instance दूसरी entity के रूप से ज्यादा instance से जुड़ा होता है, तो उसे one to many relationship कहा जाता है। जैसे एक customer बहुत सारे orders दे सकता है परन्तु एक order बहुत सारेcustomers के लिए नहीं हो सकता।



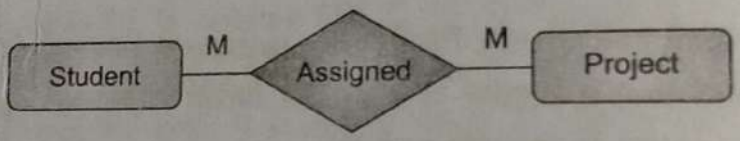
3. Many to One Relationship

जब किसी entity के बहुत सारे instance किसी दूसरी entity के केवल एक instance से संबंधित होती है, तो उसे many to one relationship कहते है। जैसे बहुत सारे students एक college से संबंधित होते है। परन्तु एक student एक साथ बहुत colleges में नहीं पढ़ सकता।



4. Many to Many Relationship

जब किसी entity के एक से ज्यादा instances दूसरी entity के एक से ज्यादा instance से संबंधित हो तो इसे many to many relationship कहा जाता है। जैसे बहुत सारे students को एक से ज्यादा projects दिए जा सकते है।

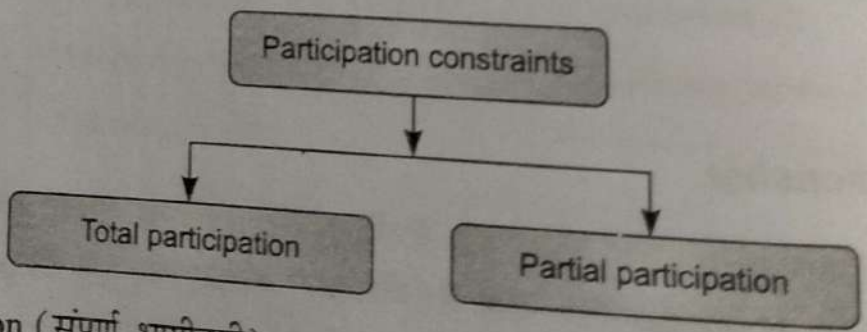


2.8 PARTICIPATION CONSTRAINTS

(Relationship में भागीदारी के नियम) Participation constraint यह बताता है कि किसी entity को relationship में भागीदारी करने के लिए कितने कम से कम instance की जरूरत पड़ेगी।

Types of Participation Constraints

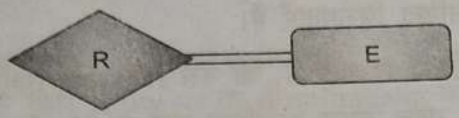
Participation constraint दो प्रकार के होते है:



1. Total participation (संपूर्ण भागीदारी)
2. Partial participation (आंशिक भागीदारी)

1. Total Participation:

- यह बाताता है कि entity set की प्रत्येक entity को कम से कम एक relationship instance में भाग लेना जरूरी है।
- इसलिए इसे mandatory participation (जरूरी) भी कहा जाता है।
- Total participation को entity और relationship के बीच में double line से दिखाया जाता है।



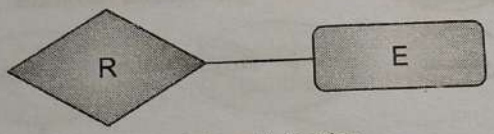
Total participation

उदाहरण:

- यहाँ entity set student और relationship set 'Enrolled in' के बीच की double line यह बताती है कि यहाँ total participation है।
- यह बताता है कि प्रत्येक student ने किसी ना किसी course में enroll किया हुआ है।

2. Partial Participation

- यहाँ जरूरी नहीं है कि entity set की प्रत्येक entity relationship में भागीदारी करें।
- इसलिए इसे एच्छिक (optional) participation भी कहा जाता है।
- Partial participation को entity set और relation set के बीच में single line से दिखाया जाता है।



Partial participation

उदाहरण:

- यहाँ single line partial participation को दिखाती है।
- इसका मतलब है कुछ student ऐसे भी हो सकते है जिन्होंने कोई भी courses नहीं लिया है।

Cardinality और Participation Constraints के बीच संबंध

Minimum cardinality बताती है कि participation total है या partial

- अगर minimum cardinality = 0, तो participation partial है
- अगर minimum cardinality = 1, तो participation total है।

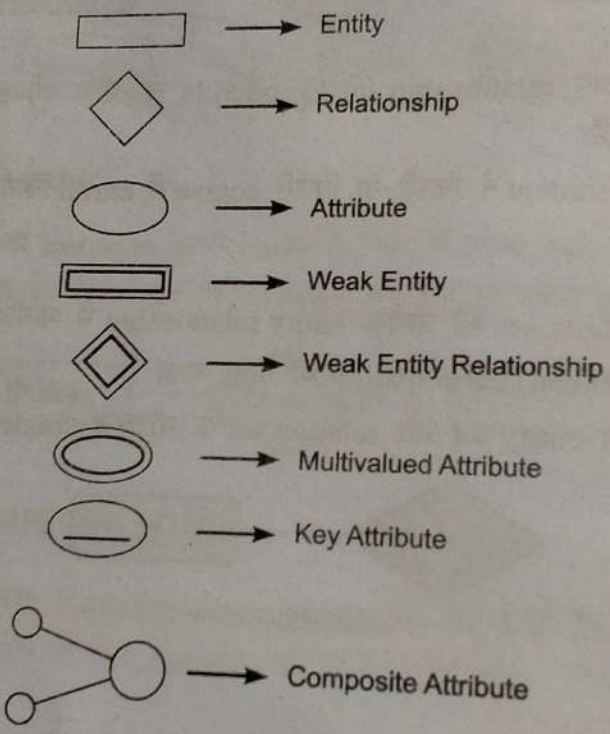
Maximum cardinality यह बताती है कि अधिकतम कितनी entities ने relationship set में भागेदारी की है। (participates किया है)

2.9 E-R DIAGRAM NOTATIONS

E-R diagrams को real-world के objects जैसे person, car, company इत्यादि को दिखाने के लिए इस्तेमाल किया जाता है। एक E-R diagram में निम्नलिखित विशेषताएँ है।

- यह database के E-R model को दिखाता है, जो उसे relations (tables) में बदलने में आसान होता है।
- E-R diagrams के लिए कोई बहुत ज्यादा तकनीकी ज्ञान या Hardware की आवयकता नहीं होती।
- यह किसी नए user के लिए समझना और बनाना बहुत असान होता है।
- यह data को logically देखने के लिए एक standard समाधान देता है।

E-R diagrams की Notations :



2.10 STRONG ENTITY SET AND WEAK ENTITY SET

वह entity set जिसमें primary key के लिए पर्याप्त attribute नहीं होते उसे weak entity set कहा जाता है। एक entity set जिसमें primary key होती है, उसे strong entity set कहा जाता है। मान लिजिए एक entity set payment है, जिसमें तीन attributes है: payment_number, payment_date and payment_amount. हालांकि प्रत्येक payment entity अलग-अलग है तब भी अलग-अलग loans की payment के लिए एक ही payments number हो सकता है। इसलिए इस entity set में कोई primary key नहीं है, और यह एक weak entity set है। प्रत्येक weak set एक one-to-many relationship set का हिस्सा होता है।

एक strong entity set के सदस्य को dominant entity कहा जाता है और weak entity set के सदस्य को subordinate entity कहा जाता है। एक weak entity set में primary key ही नहीं होती परन्तु हमें उन entries को अलग-अलग पहचान करने की आवयश्कता होती है जो एक strong entity set पर निर्भर करती है। एक weak entity set में discriminator (अलग-अलग करने वाला), attributes का set होता है, जो entity को अलग-अलग दिखा सकता है। उदाहरण के लिए payment-number एक discriminator हो सकता है। इसे entity set की partial key भी कहा जाता है।

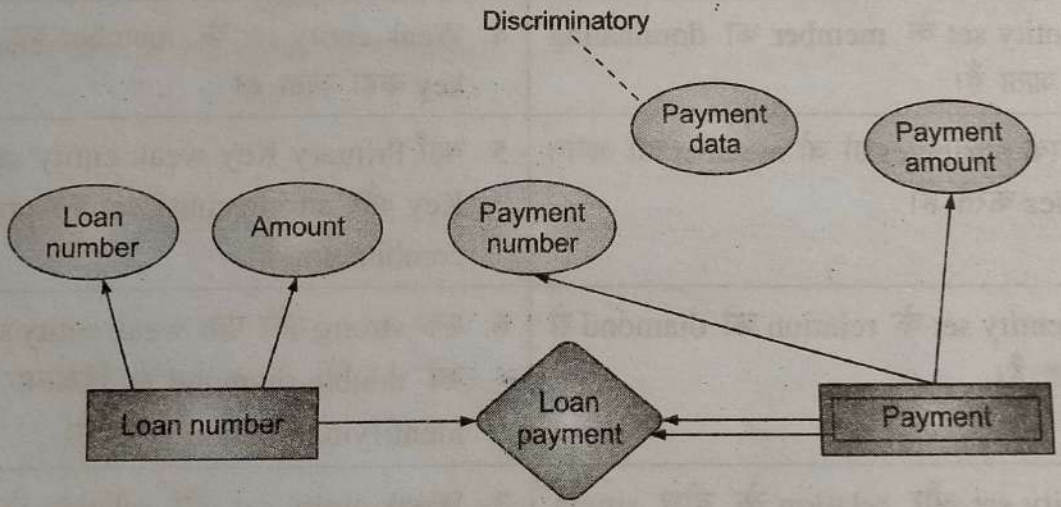
Weak entity set की primary key उस strong entity set की primary key से बनती है जिस पर weak entity set निर्भर करता है और उसके साथ entity set के discriminator को जोड़ा जाता है। उपर दिए गए उदाहरण में {loan_number, payment_number} primary key होगी।

Strong entity और weak entity के बीच के संबंध को identifying relationship भी कहा जाता है। इस उदाहरण में loan, payment entity के लिए identifying relation है। Weak entity set को doubly outline box से दिखाा जाता है और उसकी identifying relationship को double outline diamond से दिखाया जाता है। जैसा चित्र में दिखाया गया है।

यहाँ double line weak entity के total participation को दिखा रही है, जिसका मतलब है कि प्रत्येक payment entity किसी न किसी loan payment relation से जुड़ी हुई हैं। weak entity के discriminator को dashed lines से underline किया गया है।

Strong और weak entity के बीच संबंध

Relation Between strong and weak entity set

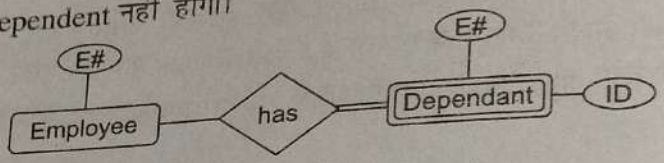


हम एक और scenario लेकर चलते है, जिसमें employees और उसके dependents की information store की गई है। प्रत्येक employee के लिए o से n तक कितने भी dependent हो सकते है। प्रत्येक dependent की एक ID और एक नाम होता है।

अब एक database को consider करते है:

- तीन employees है जिनका employee no. क्रमश: 1, 2, 3 है।
- E # 1 के दो dependent है : 1. Rahat, 2. Chahat.
- E# 2 के भी दो dependents है।
- E# 3 के तीन dependents है : 1. Raju, 2, Ruhi, 3 Raja.

इस case में dependent entity Id primary key नहीं हो सकती क्योंकि वह अलग-अलग नहीं है। तो dependent एक weak entity है, जिसका discriminator Id है। इसका relationship 'has' के साथ total participation है, क्योंकि बिना employee के कोई भी dependent नहीं होगा।



मतलब प्रत्येक Dependent के लिए कोई न कोई employee जरूर है।

इस E-R diagram से दो table बनेगी एक employee की जिसमें employee no. (E#) primary key होगा और दूसरी Dependent की जिसमें E# भी होगा और (Dependent की) ID भी होगी। इस table की primary key (E# ,ID) होगी।

तुलना:

Strong Entity Set	Weak Entity Set
1. इसकी एक primary key होती है।	1. इसके पास अपनी primary key नहीं होती।
2. इसे rectangle से दिखाया जाता है।	2. इसे double rectangle से दिखाया जाता है।
3. इसमें एक primary key होती है जिसे underline से दिखाया जाता है।	3. इसमें एक discriminator होता है जिसे dashed underline से दिखाया जाता है।
4. Strong entity set के member को dominating key कहा जाता है।	4. Weak entity set के member को subordinate key कहा जाता है।
5. Primary एक attributes हो जो member को अलग से identifies करते है।	5. यहाँ Primary Key weak entity set की partial Key और strong entity set की primary key का combination है।
6. दो strong entity set के relation को diamond से दिखाया जाता है।	6. एक strong और एक weak entity set के relation को double diamond से दिखाया जाता है, जिसे identifying relation कहते है।
7. Strong entity set और relation के बीच single line होती है।	7. Weak entity set और relation के बीच double line होती है।
8. Total participation हो भी सकता है और नहीं भी हो सकता।	8. Identifying relationship में total participation होना जरूरी है।

अभ्यास प्रश्न

1. File management system के बारे में बताइए।
2. E-R diagram के बारे में बताइए।
3. Hierarchical Model के बारे में बताइए।
4. Relational Model के बारे में बताइए।
5. Network Model के बारे में बताइए।
6. अलग-अलग तरह की keys के बारे में बताइए।
7. E-R diagram के लाभ और हानि लिखिए।
8. E-R diagram को Notations लिखिए।
9. Participation constraint को विस्तार से समझाइए।.
10. Mapping constraint के बारे में बताइए।
11. Generalization के बारे में बताइए।
12. Weak और strong entity set की तुलना कीजिए।
13. Weak और strong entity set आपस में कैसे संबंधित होते हैं।
14. Network model के बारे में detail में बताइए।
15. अलग-अलग तरह के attributes को समझाइए।

❑❑❑

3

रिलेशन मॉडल (RELATION MODEL)

3.1 RELATIONAL DATA MODEL

Relational data model को पूरे विश्व में Data storage और उसकी Processing (Data का संशोधन) के लिए मुख्य रूप से इस्तेमाल किया जाता है। कि यह Model बहुत ही सरल है और इसमें Data को efficiently (प्रभावशाली तरीके से/कुशलतापूर्वक) store करने के लिए सभी क्षमताएँ और गुण है।

Concepts:

- **Tables:** Relational data model में सम्बंध Tables (तालिका/सारिणी) के रूप में Store होते है। यह format entities के बीच के संबंधों (reltation) को store करता है। एक Table, Rows और Column का एक समूह होता है, जहाँ row, records प्रस्तुत करती है और column किसी entity के attributes (Properties) को प्रस्तुत करता है।
- **Tuple:** Table की प्रत्येक Row को tuple कहते है। जिसमें केवल एक ही Record होता है।
- **Relation Instance:** Tuple के एक finite set (परिभाषित/Set) को RDBMS (Relational data base system) में Relation instance कहते है। इसमें एक जैसे दो Tuple नहीं हो सकते।
- **Relation Schema:** Relation Schema वह है जो एक टेबल (table) के नाम के साथ उसके attributes प्रस्तुत करती है। जैसे CUSTOMER (ID, Name, Address) 'Customer Table' के लिए एक relation schema है।
- **Relation Key:** किसी एक Row के एक या अधिक से attributes, जिनसे पूरे Relation table में Rows को आसानी से अलग किया जा सके या पहचाना जा सके, उन्हे Relation key कहते हैं।
- **Attribute domain:** एक attribute के लिए पहले से कुछ values defines होती है जिन्हें attribute की domain कहा जाता है।

Constraints

प्रत्येक relation की कुछ conditions होती है, जो एक valid relation स्थापित करने के लिए पूरी होनी जरूरी है, जिन्हे Relational Integrity Contraints कहते है। यह तीन प्रकार के होते हैं–

- Key constraints
- Domain constraints
- Referential integrity constraints

Key Constraints

Relation में attributes का एक कम से कम एक ऐसा subject (समूह) होना चाहिए जो एक tuple या row को विशेष रूप से Identify कर सके। इसी subset को उस Relations का key कहते है। अगर इस प्रकार इस के और भी subsets हैं तो उन्हे Condidate key कहा जाता है।

Key constraints यह सुनिश्चित करने है कि:

- किसी relation में कोई भी दो tuble या row ऐसे नहीं होने चाहिए जिनके key attributes एक जैसे हो। जैसे कि एक Roll no के दो student नहीं होने चाहिए।
- key attribute की value null (कुछ भी नहीं) खाली नहीं होनी चाहिये। ऐसे विद्यार्थी का Record नहीं बन सकता जिसका Roll no ही ना हों अगर उस table में Roll no ही key attribute है तो।

Domain Constraints

Attributes की values ऐसी होनी चाहिए जो वास्तविक हो या असल में exits करती हो जैसे कि अगर age एक attribute है तो वह positive संख्या ही हो सकती है। Negative संख्या 'age' attribute की Domain में नहीं हो सकती। इसी तरह किसी Relation के सभी attributes पर वह Constraint लागू होता है।

Referential integrity Constraints

यह Foreign key के concept पर काम करता हैं यह दो Tables के मध्य में लिंक होता है।

यह contraintte यह बताता है कि अगर दो एक जैसे या अलग-अलग relations में किसी key attribute के द्वारा सम्बन्ध स्थापित है तो key element का होना आवश्यक है। जैसे अगर cutomer की दो tables है: एक customer-detailes है और दूसरी customer-bill है तो दोनों में अगर key attribute Customer-ID है तो प्रत्येक Record के लिए customer ID होना जरूरी है।

फायदे:

- **Simplicity:** Relational data model hierarchical और network model की अपेक्षा बहुत सरल होता है।
- **Structural Independence:** इस तरह के Model में सिर्फ Data से मतलब होता है, structure से उसका कोई मतलब नहीं होता। यह Model की performance (प्रदर्शन) को बढ़ाता है।
- **Easy to use:** यह इस्तेमाल करने में बहुत आसान है, क्योंकि Data tables में Rows और columns के रूप में होता है, और समझने में आसान होता है।
- **Query capability:** अगर data से Report बनाने के लिए कोई query लिखी जाए, तो वही आसान होता है।
- **Data independence:** Database का Structure कभी भी बिना application में बदलाव किए Change किया जा सकता है।

- **Scalable:** इस तरह के Database में records, row, or Fileds की संख्या को बढ़ाया जा सकता है जिससे इसकी उपयोगिता बढ़ जाती है।

Disadvantages *(नुकसान)*

- कुछ Databases में field की lengths of बढ़ाया नही जा सकता।
- जब कभी Data बहुत ज्यादा हो जाता है तो Relational database की जटिलता बढ़ जाती है और Data के बीच के relations भी उलझ जाते हैं।
- इस तरह के जटिल relational database से information को एक system से दूसरे में transfer करने में काफी मुश्किल होती है।

3.2 CODD के 12 नियम

Codd's 12 Rules (Codd's के 12 नियम) एक तेरह नियमों का समूह है (0 से लेकर 12 तक) जिसे Edgeer F.Codd, जो कि Relational Model के महान अन्वेषक थे, ने सुझाया है। इसमें Design से लेकर Define (परिभाषित करना) तक हर चीज बताई गई जो एक Database Mangement System को Relational Databse management System (RDBMS) बनाने के लिए आवश्यक है। Codd ने ये नियम इसलिए बताए, ताकि Relational databse को लेकर जो सपना उन्होंने देखा था उसका उद्देश्य खत्म ना हो जाए। क्योंकि Relational database की लोकप्रियता के कारण 1980 में कुछ Vendors (विक्रेता) पुराने database system को ही Relational database बताकर पेश कर रहे थें।

नियम न० 12 इसी तरह की चीजों से निपटने के लिये बनाया गया था।

नियम

Rule (0): जो System तीनों जरूरतों को पूरा करे:

Relational की, एक database की और एक Management system की, एक system तभी RDBMS कहलाएगा जब वह relational (सम्बंधो) की (Functionality) को Database को Management करने में इस्तेमाल करेगा।

Rule (1): सूचना का नियम *Information Rule*:

Database में सारी सुचनाएँ एक ही तरीके से store होनी चाजिए जैसे Values Tables में Rows और Columns के रूप में होनी चाहिए।

Rule (2): *Guaranteed Access Rule:*

सारा Data सुलभ (acessible) होना चाहिए। यह नियम Primary key की जरूरत का निश्चित करता है। यह बताता है कि कोई भी Value (Data), table के नाम, प्राइमरी key तथा column के आधार पर access की जा सके।

Rule (3): *Systematic Treatement of null Values:*

Database में Record enter करते समय हम कई values को छोड़ देते है, जिनकी या तो डालने की जरूरत नहीं होती या उस समय गलती से छूट जाती है। इन्हें System में 'Null Values' की तरह माना जाए, यह ना तो 'Zero' होती है और ना ही Blank Space. 'Null Values' को Data base को systematic तरीके से treat करना चाहिए।

Rule (4): *Active online Cetalogue based on the relation model:*

Relation Database system में एसी Catelogue (सूची) होना चाहिए जिसे online, inline किसी भी authorize user के द्वारा सामान्य Query Language से Access किया जा सके। मतलब जो query Language database के data को access करने के लिए इस्तेमाल की जाती है, उसी के द्वारा User database का structure access कर सकें।

Rule 5: *The comprehensive data sub-language rule:*

प्रत्येक Database में कम से कम एक Query Language का Support होना आवश्यक है।

1. जिसका Simple Syntax हो (आसान वाक्य संरचना हो)।
2. जो application program में भी Use की जा सके और User और Database के बीच Interaction में भी काम आए।
3. यह Language Data Define Operations, View Operations, Data की सुरक्षा Integrity (अखंडता) और Transaction Management Operations (जैसे begin, commit और rollback) को Support करें।

Rule 6: *The view updating rule:*

RDBMS में के किसी table के logical representation के तरीके को view कहते है। प्रत्येक view undatable (update करने योग्य) हो system के द्वारा updatable होने चाहिए।

Rule 7: *High-level insert, update, और delete;*

Database में Insert update और Delete functions को सिर्फ प्रत्येक group पर लागू किया, जाना चाहिए सिर्फ एक Single Row या Single table पर नहीं।

Rule 8: *Physical data independence:*

Data के Physical Level (Data कैसे store किया गया है) में बदलाव होने से, application structure में बदलाव की जरूरत नहीं होनी चाहिए।

Rule 9: *Logical data independence:*

अगर Data के Logical level में बदलाव हो जैसे कि table में, Row या Column में, तो application structure में बदलाव की जरूरत नहीं होनी चाहिए। Logical data Independence को हासिल करना Physical data independence से ज्यादा मुश्किल होता है।

Rule 10: *Integrity independence:*

Input किए गए डाटा की Integrity बनाए रखने के लिए Integrity constraints होना जरूरी है। Integrity Constraint application Program से अलग होनी चाहिए और Catalog में store किया जाना चाहिए। जब भी ऐसे constraints में बदलाव की जरूरत हो, तो application पर प्रभाव नहीं पड़ना चाहिए।

Rule 11: *Distribution independence:*

Data base में काम करते समय user को नहीं पता चलना चाहिए की database अलग-अलग जगह पर Distributed है या नहीं। मौजूदा application सही प्रकार से चलती रहनी चाहिए–

1. जब DBMS का distributed Version पहली बार Introduce किया गया और
2. जब मौजूदा distributed data को दोबारा redistribute किया जाए।

Rule 12: *The nonsubversion rule:*

अगर system कोई low-level-Interface Provide करता है तो यह system को बदलने के लिए इस्तेमाल नहीं किया जाना चाहिए, उदाहरण के लिए, किसी relational security या Integrity constraint को by-pass(Ignore) करना।

3.3 RELATIONS

किसी भी Relational database में Database relation का मतलब प्रत्येक table से है। Table को relation इसलिए कहा जाता है क्योंकि यह Data के बीच के Relation (संबंध) को Column-row के रूप में store रखती है। जहाँ column table के attributes है और row data के records है।

Relation की औपचारिक परिभाषा

Relationship किसी object-oriented model के लिए भी उतनी ही जरूरी है जितनी की relational model के लिए। नीचे relationships के बारे में कुछ महत्त्वपूर्ण तथ्य बताए गए है, जिन्हे Schema design करते समय ध्यान में रखना जरूरी है। किसी भी relational model में, relational table का समूह की Database कहलाता है। एक relational table एक 'Flat file' होती है जो columns के set से बनी होती है और इसमें और आवश्यकता के हिसाब से Rows होती है। Rows के अन्दर वही सूचनाएँ बार-बार आती है जो table के बारे में बताती है। इनके attributes (गुण) समान होते है पर value अलग-अलग होती है, key values का अलग होना जरूरी है। Row और Column जहाँ पर मिलते है उसे Cell कहते है जिसमें Data value store हाते है। हर named column की अपनी एक Domain होती है, जो कुछ values का set होता है, जो column में दिखाई देती है। नीचे दिए गए चित्र में एक relational table दिखाई गई है जिसमें book tittle, authors और publishers के बारे में information store की गई है

एक Relational Data Base

AUTHOR

au_id	*au_lname*	*au_fname*	*address*	*city*	*state*
172-32-1176	White	Johnson	10932 Bigge Rd.	Menlo Park	CA
213-46-8915	Green	Marjorie	309 63rd SI. #411	Oakland	CA
238-95-7766	Carson	Cheryl	589 Darwin Ln	Berkeley	CA
267-41-2394	O'Leary	Michael	22 Cleveland Av. #14	San Jose	CA
274-80-9391	Straight	Dean	5420 College Av.	Oakland	CA
341-22-1782	Smith	Meander	10 Mississippi Dr.	Lawrence	KS
409-56-7008	Bennet	Abraham	6223 Bateman St.	Berkeley	CA

427-17-2319	Dull	Ann	3410 Blonde St.	Palo Alto	CA
472-27-2349	Gringlesby	Burt	PO Box 792	Covelo	CA
486-29-1786	Locksley	Charlene	18 Broadway Av.	San Francisco	CA

TITLE

title id	*title*	*type*	*price*	*pub_Id*
BU1032	The Busy Executive's Database Guide	business	19.99	1389
BU1111	Cooking with Computers	business	11.95	1389
BU2075	You Can Combat Computer Stress!	business	2.99	736
BU7832	Straight Talk About Computers	business	19,99	1389
MC2222	Silicon Valley Gastronomic Treats	mod_cook	19.99	877
MC3021	The Gourmet Microwave	mod_cook	2.99	877
MC3026	The Psychology of Computer Cooking	UNDECIDED		877
PC 1035	But is it User Friendly?	popular_comp	22.95	1389
PC8888	Secrets of Silicon Valley	popular_comp	20	1389
PC9999	Net Etiquette	popular_comp	1389	
PS2091	Is Anger the Enemy?	psychology	10.95	736

PUBLISHER

pub_id	*pub_name*	*city*
736	New Moon Books	Boston
877	Binnel & Hardley	Washington
1389	Algodata Infosystems	Berkeley
1622	Five Lakes Publishing	Chicago
1756	Ramona Publishers	Dallas
9901	GGG&G	Munchen
9952	Scootney Books	New York

PUBLISHER

au_id	*ttle_id*
172-32-1176	PS3333
213-46-8915	BU1032
213-46-8915	BU2075
238-95-7766	PC1035
267-41-2394	BU1111
267-41-2394	TC7777
274-80-9391	BU7832

9999	Lucerne Publishing	Paris

409-56-7008	BU1032
427-17-2319	PC8888
472-27-2349	TC7777

Relational tables के बारे में बताने के लिए वैकल्पिक नामों का इस्तेमाल किया जाता है कुछ Manual relational tables, column और rows के लिए tables, fields और records जैसे शब्दों का इस्तेमाल करते है। औरपचारिक रूप से देखा जाए तो Relational tables column और row के लिए परस्पर relations, attributes और tuples का उपयोग किया जाता है।

इस Document में	Formal रूप में	बहुत से Database में
Relational	Table Relation	Table
Column	Attribute	Field
Row	Tuple	Record

Relational tables को sample data के बिना केवल table के नाम और column के नाम से भी दशार्या जा सकता है। **उदाहरण के लिए:**

AUTHOR

(au_id, au_lname, au_fname, address, city, state, zip)

TITLE

(title_id, title, type, price, pub_id)

PUBLISHER

(pub_id, pub_name, city)

AUTHOR_TITLE

(au_id, title_id)

Relational Data base के गुण (Properties)

Relation table की छह: Properties है:

- **Values Are Atomic:** इस property का मतलब है कि table के किसी भी column में group या array repeated नहीं होने चाहिए। इस तरह के Table को "First normal form" (1NF) कहते है। यह property बहुत जरूरी है क्योंकि यह Relational Model की एक आधारशिला है। इससे Data manipulation logic आसान होता है।
- **Column Values एक तरह की होनी चाहिए:** एक column की सभी values एक ही Domain से होनी चाहिए। Domain किसी भी column (attribute) की सभी Possible (संभावित) values का set है।

उदाहरण के लिए कोई column अगर monthly salary का है तो उसमें Monthly Salary का Data ही enter होना चाहिए, ना कि कोई और सूचना जैसे कि comments कोई status और ना ही weekly salary.

- **Relational Model 93:** यह property सुनिश्चित करती है, कि relational table की कोई भी दो row एक समान नहीं होनी चाहिए। कोई एक column या set of column (colum का समूह) ऐसा होना चाहिए, जिसकी values उस row को अलग से Define (परिभाषित) कर सके। इन्हें Primary keys कहते हैं।
- Column के sequence से कोई फर्क नहीं पड़ना चाहिए (The sequence of column is insignificant) इसका मतलब Relational table is column किसी भी न० पर हो, वही से दोबारा Use (Retrieve) किया जा सके। इसका फायदा यह है कि User किसी भी tables को यह ध्यान में रखे बिना share/इस्तेमाल कर सकते हैं कि वह किस तरह organised है। यह relational table को बदले बिना physical structure को बदलने की सुविधा भी प्रदान करता है।
- **The Sequence of rows is Insignificant:** Row भी किसी न० पर हो कोई फर्क नहीं पड़ता इसका फायदा भी यहीं है कि Rows किसी भी क्रम में Retreive की जा सकती है। Relational table में information add करना आसान हो जाता है और यह मौजूदा Queries के भी प्रभावित नहीं करता।
- **प्रत्येक Row अलग होनी चाहिए :** यह property सुनिश्चित करती है कि, relational table की कोई भी दो row एक समान नहीं होनी चाहिए।
- प्रत्येक column का एक अलग नाम होना चाहिए। चूंकि column के न० से कोई फर्क नहीं पड़ता, column की पहचान उसके नाम से होनी चाहिए। मतलब, column का नाम Database में अलग होना चाहिए, परन्तु उसी table से होना चाहिए जिससे वह belong (संबंधित) करता है।

3.4 SCHEMA AND SUB-SCHEMA

Database Schema औपचारिक भाषा मे describe किया हुआ databse system का structrue है, जो data का संगठन है जो यह बताता है कि data tables मे किस प्रकार बाटाँ गया है। Database schema की औपचारिक परिभाषा यह है कि यह एक Formulaes का set है जो database में Integrity constants को लागू करता है। सभी Constraints एक ही भाषा में होते है। Conceptual schema की अवस्थाओं (states) को explicit mapping करके database schema में बदला जाता है। यह बताता है कि Real World की entities को database के रूप में कैसे Modeled किया जाता है। Database designer अपनी application की समझ से database schema को design करते है, जिसे programmer आसानी से database को समझ और इस्तेमाल कर सके। Database schema की धारणा वही काम करती है, जो Predicate calculus की धारणा करती है। मतलब इसकी theory Database से इस तरह मेल खाती है कि किसी भी समय इसे Mathematical Object (गणितीय वस्तु) के रूप में देखा जा सकता है। इसलिए Schema में बहुत सारे Formulas होते है, जो किसी application के लिए (या database के type के लिए) Integrity constraint को represent करते है, Relational database में schema tables fields, relationships, views, procedures, functions, queries, triggers types, sequences materialized vies, synonyms database links, directories, Java, XML आदि को परिभाषित करती है।

Schema साधारणतया: Data directory में stored होती है। हाँलाकि Schema को text lagnuage में define किया जाता है, इसे Database के graphical चित्रण को बताने के लिए प्रयोग किया जाता है, ये भी कह सकते है कि schema database का structure है जो उसके objects को define करती है।

Database Schema के स्तर (Levels)

- **Conceptual schema:** यह concept और उनके relationships की Mapping (चित्रण) करती है।
- **Logical Schema:** यह entities और उनके attributes और relation का चित्रण करती है।
- **Physical schema:** यह logical schema का एक विशेष रूप से implementation है।
- **Schema object:** यह oracle database object है।

सभी database relationship schema में पहले से ही परिभाषित (predefined) होते है। इसका मतलब की सभी records types और set types जो DBMS/R database network बनाने में काम आते है की परिभाषा को schema कहते है।

यह DBMS द्वारा offer/use की गई Language की schema का formulation है। इस formulation के परिणाम से, न केवल scheme और conceptional schema ही represent किया जा सकता है, अपितु conceptual data को create (बनाने) और maintain करने के नियम भी स्थापित किए जा सकते है, इन दोनों चीजों को करने के लिए schema को दो भागों में बाटाँ जाता है। जैसे कि:

- A logical schema
- A physical schema

एक **Logica schema** वह है जो DBMS द्वारा offer किये गए data structure को संशोधित करके उसे computer द्वारा समझने के योग्य बनाता है।

जबकि **Physical schema** वह है जो, यह देखता है कि conceptual databse को computer में किस तरह से store database के रूप में represent किया जा सके।

एक **Logica schema** specific problems domain का एक data model जिसे data managemtn techonology के शब्दो में भी express किया जा सकता है।

किसी एक Data Management Product को विशिष्टता दिए बिना, ये या तो relational table और column, object oriented class या XML tags की तरह हो सकता है। यह conceptual data model का उलट रूप है, जो किसी organisation के अर्थ को बिना technology के reference के समझाता है, या एक Physical data model से भी मेल नही खाता जो data को storage medium में store करने के physical mechanism को समझाता है।

Logical और physical schema दोनों की properties (गुणों) को DBA के लिए आसान बनाने के लिए DBMS (database management system) दो तरह की भाषाओं का इस्तेमाल करता है, Data defination Language (DDL) और Data storage description Language (DSDL). यह उस तरीके को control करने की सुविधा प्रदान करता है जिससे Data structure को computer में दिखाया जाता हैं। इसमें, storage space को allocate करना, access methods का चुनाव करना, वह विशेष तरीका जिससे अलग-अलग objects एक-दूसरे से संबंधित है, शामिल है। इसकी efficiency (दक्षता) को समय और space दोनो की बचत से मापा जा सकता है।

Physical Schema: यह data management से संबंधित है।

ANSI four-schema architecture, में Internal schema डाटा का वह view है जिसमें Data Management की तकनीक होती है। इससे विपरित external schema organisation के प्रत्येक व्यक्ति का view दर्शाता है, और conceptual schema external schemas के एकीकरण (Integration) का एक set है। Internal schema: को भी दो भागों में बाँटा जाता है:

Logical Schema वह तरीका है जिससे Data को represent किया जाता है, किसी भी database Management के constraints के अनुसार।

इस समय पर hierarchical और Network के विकल्प होते है। Logical Schema के बारे में बताने से; यह नहीं पता चलता की data को disk driver में Physically कैसे store किया जाता है। यह physical schema का काम है। अब Logical schema data को relational tables और columns, object oriented classes, और XML tags के रूप में वर्णित (Describe) करता है। Tables का एक Set बहुत अलग-अलग तरीकों से Use किया जा सकता है, जिसमें architecture भी शामिल है, जैसे कुछ row किसी computer में होती है, जो एक स्थान पर है और कुछ rows किसी दूसरे computer में होती है जो दूसरे स्थान पर है। संबंधित Language translator के द्व रा Logical और physical schema को process करने के बाद, computer में एक Null Database बन जाता है। (जिसमें structure और Data को store करने के तरीके होते है और जब Data डाला जाता है, तो वह उसी structure और Logic के हिसाब से काम करता है।)

यह बिना Data का एक Database होता है। मतलब की Conceptual database के अनुरूप Database computer में load कर दिया जाता है, इस database को conceptual database में कोई भी बदलाव होने पर operate किया जाता है।

CONCEPTUAL SCHEMA

Conceptual schema या conceptual data model मॉडल Concept और उनके संबंधो का एक चित्रण है। यह किसी organisation के semantics (अर्थ विज्ञान) के बारे में बताता है, उसकी प्रकृति के बारे में बताता है। यह किसी organisation की महत्त्वपूर्ण चीजे बताता है जैसे (entity classes), जिनके बारे में ये सूचना इकट्ठा करता है, उनकी विशेषताओं (attributes), और उनके बीच के सम्बन्धों (relationalships) को भी बताता है।

OVERVIEW (अवलोकन)

क्योंकि Conceptual Schema Organisation के, Semantics (अर्थ-विज्ञान) (शब्दार्थ) को बताता है, ना कि उसके design को, यह abstraction के अनेक levels पर हो सकता है। वास्तविक ANSI Four-Schema architecture external schema के set के साथ शुरू होता है, जो किसी व्यक्ति के आसपास के बारे में view को दर्शाता है। यह सब एक Conceptional Schema में समाहित (consolidated) किया जाता है।

जो सारे external views का SUPERSET (मुख्य समूह) होता है। एक Data Model किसी व्यक्ति के दृष्टिकोण की तरह ठोस हो सकता है, परन्तु इससे वह Inflexible (लचीला नहीं होना) हो सकता है। अगर उस व्यक्ति की सोच बदल जाती है तो Database में भी बदलाव संभव होना चाहिए। Conceptual Data model abstract के दृष्टिकोण से काम करता है, मूलभूत चीजों (Fundamental things) की पहचान करके, उसमें से जिन चीजों को कई व्यक्ति deal करता है, वह सिर्फ उदाहरण है। यह Model Object Oriented terms की विरासत को अनुमति देता है। किसी entity class के उदाहरणों के समूह (Instances) को Sub-entity class उप-विभाजित में किया जा सकता है। Sub-type entity class का प्रत्येक Instance super-type (main) entity class का भी Instance होता है।

यह Super-type/sub-type relationship (संबंध) exclusive (अद्वितीय) हो भी सकती या नहीं भी। कभी-कभी ये जरूरत भी हो सकती कि किसी super-type का प्रत्येक instance सिर्फ एक sub-type का ही instance होता है। इसी तरह, Super type/sub-type relationships exhaustive (संपूर्ण) हो भी सकते हैं या नहीं। यह exhaustive

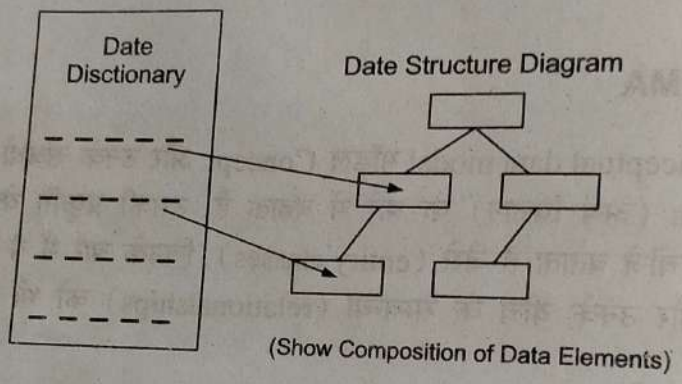
होगी अगर कार्य-प्रणाली की यह जरूरत होगी कि एक super-type का प्रत्येक Instance किसी ना किसी sub-type का भी Instance होगा।

यह तरह के सम्बंधों का उदाहरण:

- कोई PERSON एक या अधिक ORDERS Class में vendor हो सकता है।
- परन्तु प्रत्येक ORDER सिर्फ एक ही PERSON से आ सकता है।
- PERSON is sub-type of Party, (PERSON का प्रत्येक Instance party का भी Instance होगा।)
- प्रत्येक EMPLOYEE class में ही SUPERVISOR भी होगा।

Data Structure Diagram

एक data structure diagram (DSD) वह Model या diagram (चित्रण) है, जो चित्रात्मक संकेतो (Graphical notations) के द्वारा conceptual data models को describe करता है। यह document की entities, उनके बीच संबंधो और constraints को दर्शाता है।



Data Structure Diagram (Data संरचना आरेख)

एक **Conceptual Schema** सामान्यता बहुत तरह की external schema में विभाजित हो जाता है, और इससे ही Super real world view को देखता है। इसके फलस्वरूप ही DBMS में external की धारणा को capture करने के लिए कोई साधन होना चाहिए। DBMS के शब्दों में यह subschema की धारणा से संभव हो सकता है।

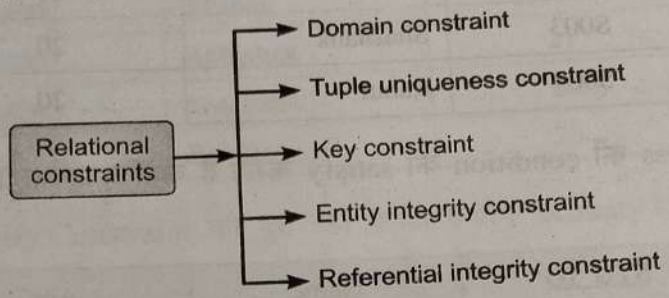
Subschema की properties external schema की properties के समान ही होती है। यह user को यह window प्रदान करता है जिससे वह Database का यही भाग देख सके जो उसकी जरूरत का हो यह property जो उसे external schema से मिली है इसके अलावा भी subschema की और भी अच्छी qualities है। यह database के controlled access (नियंत्रित पहुँच) को लागू करने के लिए भी काम करता है। उसके अलावा subschema database के concurrent operation (समवती संचालन) को भी control करता है। Subschema के लिए subschema Defination Language (SDL) को इस्तेमाल किया जाता है। इस Language का nature data के structure पर निर्भर करता है जिस पर DBMS based होता है, और host language पर भी निर्भर करता है।

Subschema: एक subschema cobol program द्वारा इस्तेमाल की जाने वाली उस schema का small view होती है, जो Data को access करने के लिए Use होती है। Subschema प्रत्येक application program के लिए database के view को simplify करने का तरीका बताती है। Subschema केवल उन्ही records types या data elements और set types को access करने की क्षमता प्रदान करती है, जिनके लिए कोई application program अधिकृत (authorised) है। Subschema user को applications के अलग-अलग भागों को access करने की सुविधा

प्रदान करता है। Application के भाग set, types, record types, data items और data aggregates होते है। Schema's में बहुत तरह की sub-schema होती है जो अलग-अलग प्रकार की होती है। Subschema's बनाने के अलग-अलग कारण हो सकते हैं। एक कारण यह भी हो सकता है कि User या Programmer अलग-अलग views देखना चाहते है, बिना database के सारे data को देखे हुए। एक और कारण यह भी हो सकता है कि सुरक्षा को बढ़ाने के लिए, कोई जो authorised (अधिकृत) नहीं है वो डाटा को change या add ना कर सके।

3.5 RELATIONAL MODEL CONSTRAINTS

Relational Model में निम्न पाँच तरह के constraints होते है।



यह एक प्रकार के नियम है जिनके पूरा ना होने पर System हम आगे नहीं बढ़ने देता।

1. Domain constraint (value domian के अन्दर होनी चाहिए।)
2. Tuple Uniqueness constraint (प्रत्येक row unique होनी चाहिए।)
3. Key constraint (Primary keys की value unique होनी चाहिए।)
4. Entity Integrity constraint
5. Referential Integrity constraint

1. DOMAIN CONSTRAINT

- यह एक attribute के लिए Domain या Values का set define करते है। जैसे हम पहले पढ़ चुके है, उदाहरण के लिए Attribute (Gender) के लिए values (Male, Female, Others) ही उसकी Domain है।
- यह सुनिश्चित करता है कि attribute के लिए enter की गई value उसकी Domain से ही हो।

Example:

नीचे दी गई Table Student को देखिए।

STU_ID	Name	Age
S001	Akshay	20
S002	Abhishek	21
S003	Shashank	20
S004	Rahul	A

यहाँ value 'A' मान्य नही है, क्योंकि age 'A' नहीं हो सकती केवल पूर्णांक ही हो सकती है।

2. TUPLE UNIQUENESS CONSTRAINT

यह constraint बताता है कि सभी rows अलग-अलग होनी चाहिए कोई भी दो row बिलकुल समान नहीं होनी चाहिए।

उदाहरण 1:

नीचे दी गई student table में:

STU_ID	Name	Age
S001	Akshay	20
S002	Abhishek	21
S003	Shashank	20
S004	Rahul	20

यह Tuple uniquer press की condition को satisfy करता है क्योंकि प्रत्येक Tuple (row) अलग-अलग है।

उदाहरण 2:

STU_ID	Name	Age
S001	Akshay	20
S001	Akshay	20
S003	Shashank	20
S004	Rahul	20

यहाँ पहले दो tuple (row) बिलकुल समान है इसलिए यह Tuple uniqueness की property को पूरा नहीं करता।

3. KEY CONSTRAINT

- ये बताता है कि किसी भी relation में, primary key की सभी values unique (अद्वितिय, अलग) होनी चाहिए।
- Primary key की value Null (खाली) नहीं होनी चाहिए।

उदाहरण के लिए नीचे दी गई Student table को देखे:

STU_ID	Name	Age
S001	Akshay	20
S001	Abhishek	21
S002	Shashank	20
S003	Rahul	20

यह key constraint के नियम को पूरा नहीं करता क्योंकि Primary keys unique नहीं है। दो students की एक ID या Roll no. नहीं हो सकता।

4. ENTITY INTEGRITY CONSTRAINT

- यह बताता है की primary key के किसी भी attribute की value null (खाली) नहीं होनी चाहिए।
- क्योंकि primary key के null होने से uniqueness property खत्म हो जाती है।

उदारहण:

नीचे दी गई STUDENT Table देखें!

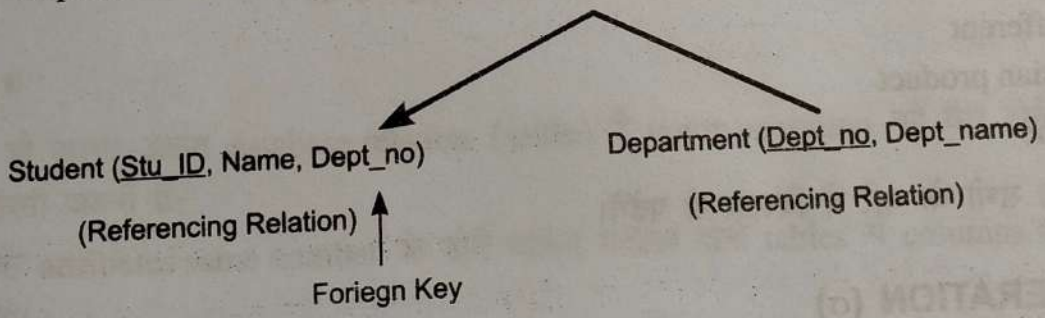
STU_ID	Name	Age
S001	Akshay	20
S002	Abhishek	21
S003	Shashank	20
	Rahul	20

यह relation entity Integrity Constraint को पूरा नहीं करता क्योंकि primary key में एक Null value है।

5. REFERENTIAL INTEGRITY CONSTRAINT

- यह constraint तब काम में आता है जब कोई foreign key किसी relation में primary key को reference (उसको संदर्भित) कर रही हो।
- यह बताता है कि foreign key की values या तो primary key के relation (table) में होनी चाहिए या Null होनी चाहिए।

उदारहण के लिए: नीचे दो Relations (table) "Student" और "Department" दिए गए है। यहाँ relation student department को refer कर रहा है।



Student

STU_ID	Name	Dept_no
S001	Akshay	D10
S002	Abhishek	D10
S003	Shashank	D11
S004	Rahul	D14

DEPARTMENT

Dept_no	Dept_name
D10	ASET
D11	ALS
D12	ASFL
D13	ASHS

यहाँ relation student referential integrity के constraint को पूरा नहीं करता।

- क्योंकि Relation Department में D14 नाम से कोई Primary key नहीं है (मतलब D14 Department Relations में है ही नहीं)
- इसलिए referential Integrity का नियम पूरा नहीं होता।

3.6 RELATIONAL ALGEBRA (BASIC OPERATION: UNION INTERSECTION DIFFERENCE AND CARTESIAN PRODUCT)

Relational algebra एक procedural language है जिसमे operators का प्रयोग Queries को Perform करने के लिए किया जाता है। इसमें relation के Instance को Input के लिए Use किया जाता है और output में भी यह Relation के Instance ही देती है। कोई Operator या तो Unary होगा या binary होगा। Relational algebra, Relation (table) पर ही इस्तेमाल होता है और उसका Result भी Relation (table) के रूप में ही होता है।

Relational algebra (बीजगणित) के कुछ fundamental (मौलिक) operations है:

- Select
- Project
- Union
- Set different
- Cartesian product
- Rename

हम आगे इन सभी के बारे में विस्तार से पढ़ेंगें।

SELECT OPERATION (σ)

यह उस Tuple (row) को Select करता है जो किसी relation में दी गई condition को satisfy करती है।

Notation: $\sigma_p(r)$

यह σ (sigma) से दर्शाया जाता है, r यहाँ relation है और p वह logical formulae है जो connector जैसे and, or और Not को Use करता है। यह सभी relational operators जैसे -: =, ≠, ≥, <, ≤, का इस्तेमाल करते है।

उदाहरण के लिए

$\sigma_{subject = \text{"...}}$ AsianPublisherBooks)

Output: यह वो tuple select करेगा जिनका subject 'mechanics' होगा।

$$\sigma_{\text{Subject = "database" and price = "450"}}$$

or = year = > "2010" (Asian Publisher Book)

Output: यह वो tuple select करेगा जहाँ subject 'database' हों 'price': 450 हो या वह किताबें जो 2010 के बाद प्रकाशित हुई हो।

PROJECT OPERATION (π)

इसे (π) pi से दर्शाया जाता है। यह एक unary operation है जो सिर्फ एक ही table में काम करता है। यह Relation (table) के attributer (column) को दी गई conditions के आधार पर select कर लेता है:

Notation – $\Pi A_1, A_2, A_n (r)$

जहाँ A_1, A_2, A_n relation r के attributer (column है।) Duplicate rows अपने आप eliminate (खत्म) हो जाती है। क्योंकि relation एक set है।

उदाहरण के लिए:

$$\Pi_{\text{subject, author}} (\text{AsianPublisherBooks})$$

यह Table (Asian Publisher Books) में से केवल दो columns 'Subject' और 'author का नाम' ही select करता है।

UNION OPERATION (∪)

यह दी गई दो relations (table) के बीच Binary Union दिखाता है और इसे नीचे दिए गए तरीके से दिखाते है:

$$r \cup s = (t) (t \in r \text{ or } t \in s)$$

Notation: r ∪ s

जहाँ r और s दो अलग-अलग database relation (table) है union operation को वैध होने के लिए निम्न conditions पूरी करनी जरूरी है:

- r और s के attributes same number के होने चाहिए मतलब दोनो tables में columns की संख्या बराबर होनी चाहिए।
- Attributes की domain compatible होनी चाहिए।
- Duplicate tuples cross अपने आप खत्म हो जानी चाहिए।

$$\Pi_{\text{author}} (\text{AsianPublisherBooks}) \cup \Pi_{\text{author}} (\text{Articles})$$

Output: उन authors के नाम दे देगा, जिन्होने या तो कोई Book लिखि है या Article लिखा है या दोनो लिखे है। परन्तु एक author का नाम एक बार ही आएगा।

SET DIFFERENCE (−)

यह उन tuple को दे देगा, जो एक relation मे है परन्तु दुसरे में नही हैं।

Notation: r − s

यह वो tuples (rows) छाँट देगा जो 'r' में तो है परन्तु 's' में नही है।

$$\Pi_{\text{author}} (\text{AsianPublisherBooks}) - \Pi_{\text{author}} (\text{Articles})$$

Output: यह उन authors के नाम देगा जिन्होंने Book लिखी है परन्तु articles नहीं लिखें।

CARTESIAN PRODUCT (×)

यह दो relations की Informations को एक में जोड देता है।

Notation: $r \times s$

जहाँ 'r' और 's' दो relations है और इसका output इस तरह से आएगा:

$$r \times s = \{qt \mid q \in r \text{ and } t \in s\}$$

$$\sigma_{\text{place = 'Delhi'}} (\text{AsianPublisherBooks} \times \text{Articles})$$

Output: यह वह relation देगा, जिसमे Delhi से छपने वाले Asian Publishers के Book और Article दोनो हो।

RENAME OPERATION (P)

Relational algebra को जो परिणाम (Result) निकलता वह भी Relation (table) होता है परन्तु उस table का कोई नाम नहीं होता, Rename operation हमे output relation का नाम रखने/बदलने की सुविधा प्रदान करता है। इसे Greek के अक्षर Rho 'ρ' से दिखाया जाता है।

Notation: ρ × (E)

जहाँ expression E का परिणाम x से Save किया गया है।

कुछ और operations भी होते है जैसे:

- Natural join
- Projection
- Division

PROJECTION (π)

यह किसी दिए हुए Relation मे से केवल Projection List को छोड़कर बाकी सार attributes (column) को Delete कर देता है। Projection method से बना Relation किसी भी दिए गए Relation का vertical sub-set होता है।

यह Duplicate value हटाकर सिर्फ जरूरी attributable को निकालने में मदद करता है। (pi) इसका symbol है जिसे relation मे से attribute को चुनने के लिए इस्तेमाल किया जाता है।

यह operation आपको किसी relation में से कुछ विशेष column निकालने की सुविधा देता है और बाकी बचे हुए columns को हटा (delete) देता है।

Join Operations

यह एक तरह का Cartesian product ही है, परन्तु इसके साथ कुछ selection criteria भी होते है।

यह ∞ से denote किया जाता है।

Join operations अलग-अलग tables की विभिन्न tuples (rows) को भी जोड़ने की अनुमति प्रदान करता है।

Join के प्रकार:

Inner Joins: *(आंतरिक Join)*

- Theta join
- EQUI join
- Natural join

Outer join:

- Left Outer Join
- Right Outer Join
- Full Outer Join

Inner Join

इसमे, सिर्फ वही tuples जुड़ती है जो दिए गए criteria (मापदंड) से Match करती है, बाकी हटा दी जाती है। अभी हम inner joint के कुछ प्रकारों के बारे में पढ़ेगें।

Theta Join:

यह Join Operation का सबसे सामान्य रूप है जिसे θ से दर्शाया जाता है।

उदाहरण के लिए:

$$A \bowtie_{\theta} B$$

θ Join Selection Criteria में किसी भी condition का इस्तेमाल कर सकता है।

जैसे

$$A \bowtie_{A.column\ 2 > B.column\ 2} (B)$$

A ⋈ A.column 2 > B.column 2 (B)	
column 1	column 2
1	2

EQUI join:

जब θ join केवल बराबरी (equivalence) condition का इस्तेमाल करे तो यह equi join बन जाता है।

उदारहण के लिए:

$$A \bowtie_{A.column\ 2 = B.column\ 2} (B)$$

A ⋈ A.column 2 = B.column 2 (B)	
column 1	column 2
1	1

यह RDBMS का एक बहुत ही मुश्किल operation है, क्योंकि इसे लागू करने मे RDBMS की Performance problem सामने आ जाती है।

NATURAL JOIN (⋈)

Natural Join तभी काम करता है जब relations (tables) के बीच कोई समान (एक जैसा) attribute (column) हो इस attribute का नाम और प्रकार एक जैसा होना चाहिए।

उदाहरण के लिए:

नीचे दी गई दो tables को देखे:

C		D	
Num	Square	Num	Cube
2	4	2	8
3	9	3	27

C ⋈ D

C ⋈ D		
Num	Square	Cube
2	4	8
3	9	27

Outer Join

Outer join में उन tuples के साथ जो Matching criteria (दिए हुए मानदंडो) को पूरा करती है, हम उन सभी या कुछ tuples को भी रखते है जो इन conditions (मानदंड़ों) को पूरा नहीं करती।

Left Outer Join (A ⟕ B)

इस Join मे, operation, left relation (table) के सभी tuples को output में रखता है। क्योंकि Right relation में कोई मिलती हुई (Mathcing) tuple नहीं है तो right relation के उस attribute की value को 'Null' कर दिया गया।

नीचे दी गई 2 tables को देखिए:

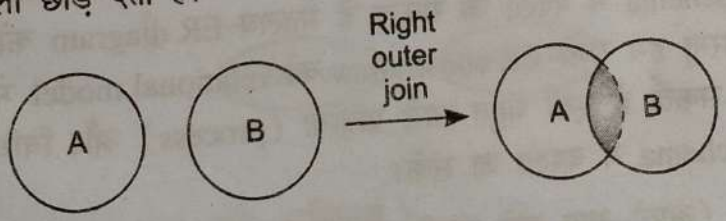
A		B	
Num	Square	Num	Cube
2	4	2	8
3	9	3	27
4	16	5	125

A ⟕ B

A ⟕ B		
Num	Square	Cube
2	4	8
3	9	27
4	16	-

Right Outer Join: (A ⟖ B)

यहाँ operation right relation (table) की अंसनम को रख लेता है और अगर left relation मे कोई matching value नही होती तो उसे खाली छोड़ देता है।



A ⟖ B

A ⟕ B		
Num	Cube	Square
2	8	4
3	27	9
5	125	—

यहाँ Right table की value को लिया गया है

Full Outer Join: (A ⟗ B)

इसमें दोनों Relations की Tuples (Rows) को Join कर दिया जाता है, बिना Mathching Condition को देखे हुए

$A \text{ ⟗ } B$

$A \bowtie B$		
Num	*Cube*	*Square*
2	4	8
3	9	18
4	16	—
5	—	75

Division Operator

Division operators A ÷ B तब ही Use होंगे जब नीचे दी गई conditions पूरी हो जाएगी।

- relations B के attributes (column) A के attributes के पूर्ण subset हों।
- Division Operator के द्वारा प्राप्त हुए relation में ये attributes होंगे:

  = (A के सभी attributes — B के सभी attribute)
- Division operator से जो result मिलेगा, उसमे relation A के वो सारे tuple होगे, जो B के प्रत्येक tuple से संबंधित होगें।

3.8 ER MODEL को RELATIONAL MODEL में बदलना

ER Model, चित्रों के माध्यम से entity-relationship के बारे में बताता है जो समझने में बहुत आसान होता है। ER diagram को relational schema में बदला जा सकता है मतलब ER diagram को देखकर relation schema (table) बनाई जा सकती है। परन्तु हम सभी ER constraints को relational model में नहीं ला सकते परन्तु एक अनुमानित schema बनाया जा सकता है एसी बहुत सारी प्रक्रिया (process) और विधियाँ (algorithm) है जिसे ER diagram को relational schema में बदला जा सके।

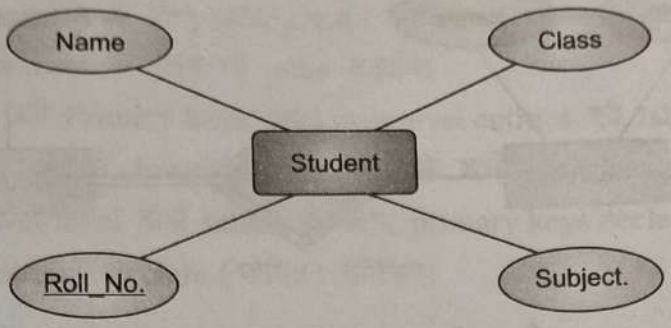
उनमे से कुछ automated (अपने आप होने वाली) है और कुछ Manual है (करनी पड़ती है।) हमे यहाँ Mapping diagram के contents (घटको) का भी ध्यान रखना पड़ेगा जो relation के basics है।

ER diagrams मुख्य रूप से इनसे बना होता है:

- Entity और attributes
- संबंध (Relationships) जो entities के बीच में है।

Mapping Entity (Entity की Mapping करना)

Entity एक वास्तविक वस्तु है जिसके कुछ गुण/property (attributes) होते है।

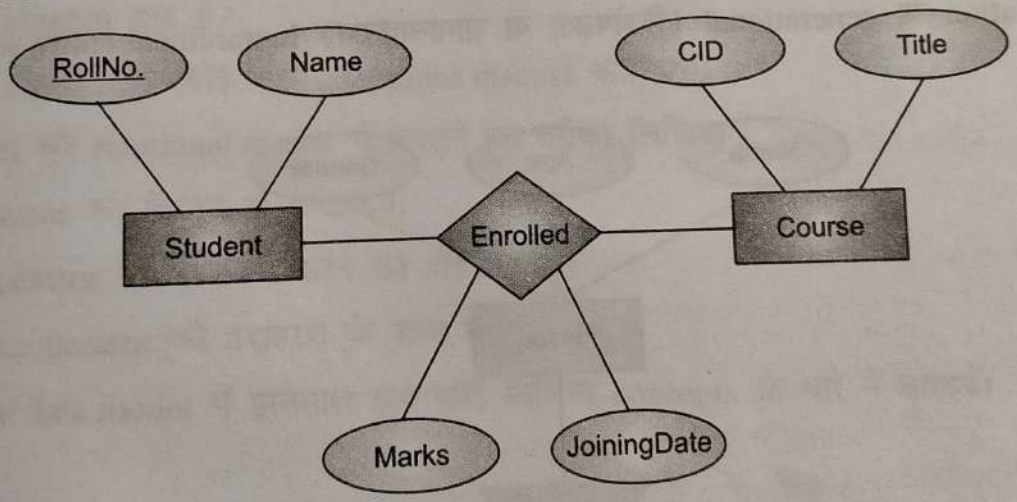


Mapping की प्रक्रिया (Algorithm)

- प्रत्येक entity. के लिए एक table बनाईये।
- Entity के attributes उस table के Fields (Columns) होने चाहिए।
- Primary key निश्चित कीजिए।

Mapping Relationship

दो या ज्यादा entities के बीच के संबंध को relationship कहते है।

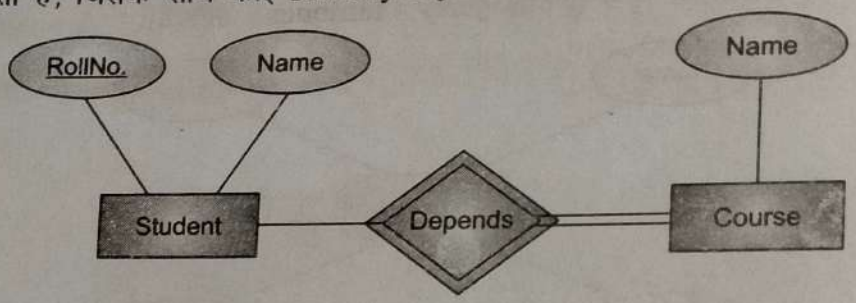


Mapping Process (Mapping की प्रक्रिया)

- किसी relationship के लिए table बनाइये।
- सभी entities की primary key को table के fields के रूप में add कीजिए उनके Data types के साथ।
- अगर relationship का भी कोई attribute है तो उसे भी table के field के रूप में add कीजिए।
- सभी entities की primary key को मिलाकर एक Primary key बनाईये।
- सभी Foreign key constraints को declare (घोषित) कीजए।

Weak entities के Sets की Mapping करना

Weak entity वह होती है, जिसके साथ कोई Primary key नहीं होती।

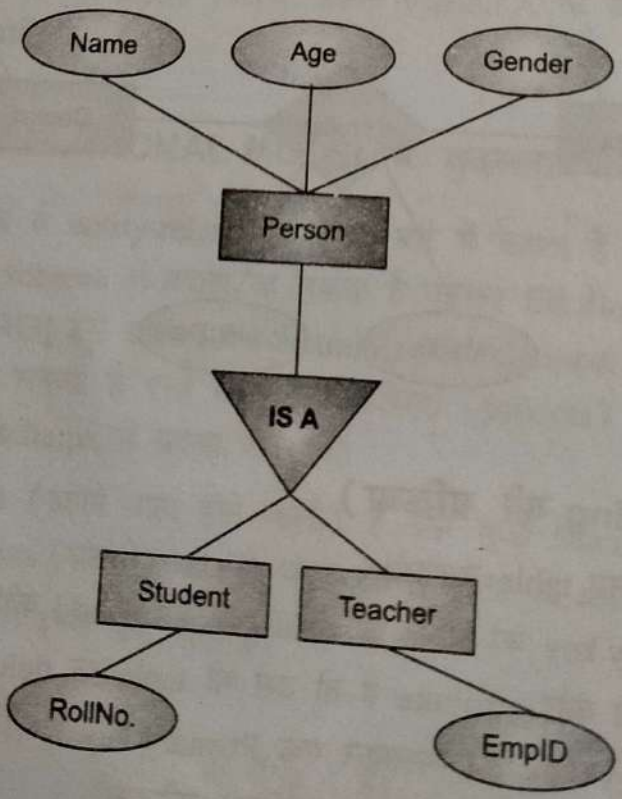


Mapping Process (Mapping की प्रक्रिया)

- Weak entity set के लिए table बनाईये।
- सभी attributes को table के field के रूप में Add कीजिए।
- Identifying entity set के लिए primary key Add कीजिए।
- Foreign key constraints को घोषित कीजिए (बताईये)।

Mapping Hierarchical Entities (पदानुक्रामित पद के हिसाब से क्रमित entities की Mapping)

ER की Specialization या generational (विशेषज्ञता या सामान्यकरण) hierarchical entity set (पदक्रमानुत) के रूप में आते हैं।



Mapping Process (Mapping की प्रक्रिया)

- सभी nigher-level entities के लिए table create कीजिए।
- सभी Lower-lever entities के लिए भी table बनाईये।
- High level entities की Primary keys को lower level entities की table में add कीजिए।
- Lower-Level की tables में, lower-level entities के सभी attributes add कर दीजिए।
- High level और lower level दोनो tables के लिए primary keys declare कीजिए।
- Foresight key constraints declare (घोषित) कीजिए।

अभ्यास प्रश्न

1. Relational model के लाभ और हानि बताइए।
2. Codd के 12 नियमों की विस्तार से समझाइये।
3. Relation की औपचारिक परिभाषा बताइये।
4. Schema और subschema पर Short Note लिखिए।
5. Relational model के constraints को समझाइये।
6. Relation algebra क्या है?
7. Relation algebra के कोई पाँच operations discuss कीजिए।
8. ER Model को relational model में बदलने का तरीका लिखिए।
9. Join Operator को विस्तार से समझाइये।
10. Divide opeartor को इस्तेमाल करने को शर्तें बताइये।
11. Projection operator को उदाहरण के साथ समझाइये।
12. Relational data model में इस्तेमाल होने वाले विभिन्न concepts के बारे में बताइयें।

❑❑❑

4

रिलेशनल डाटाबेस डिज़ाइन
(RELATIONAL DATABASE DESIGN)

4.1 NORMALIZATION का उद्देश्य

Normalisation का मुख्य उद्देश्य Duplicasy को कम करना है और Insert, Update और Delete की विसंगतियों को दूर करना है। यह बड़ी tables को छोटी tables में बाँट देता है और उन्हें relationships से link कर देता है। Data redundancy (Duplicasy) तब होती है जब एक ही तरह का Data दो अलग-अलग जगहों पर होता है। मूल रूप से Normalisation, Data को प्रभावी रूप से संगठित (organise) करने की प्रक्रिया है। Normalisation process के दो मुख्य लक्ष्य होते हैं–Duplicate data को हटाना (Same data को एक से ज्यादा table में store करके। और यह सुनिश्चित करना कि data dependencies (data की निर्भरता) को कोई मतलब है। (एक table में सिर्फ संबंधित Data ही store होना चाहिए।)

4.2 DATA REDUNDANCY AND UPDATING ANOMALIES (विसंगतियाँ)

Data Redundancy Data का दुहराव (Duplicasy) है। database में Data redundancy का मतलब है कि कुछ Data field Database में एक से ज्यादा बार entered हैं।

Data Redundancy की सामान्य परिभाषा (General definition) के हिसाब से इसे दो तरह से Classified किया जाता है-- Excessive या wasteful. Wasteful data redundancy तब होती है जब, कोई data repeat नहीं होना चाहिए परन्तु गलत coding या process के जटिल होने के कारण बार-बार repeat (duplicate) होता रहता है।

एक सकारात्मक तरह की Data Redundancy भी होती हे, जो data को सुरक्षित करती है और उसकी consistency को भी बढ़ाती है। बहुत से developer यह मानते हैं कि Data multiple places (विभिन्न जगहों) पर store होना चाहिए। इसका मकसद है कि Data के लिए एक Central (केंद्रित) Master space होना चाहिए, और ऐसा तरीका होना चाहिए कि Data को एक साथ सब जगह update किया जा सके, एक central access point के द्वारा। वरना Data redundancy बहुत बड़ी दिक्कतें पैदा कर सकती है जैसे कि Inconsistency, जहाँ एक जगह Data को update करने पर वह अपने आप दूसरी जगह update न हो। इससे जो data identical होना चाहिए था अंत में जाकर उसकी values ही अलग-अलग हो जाती हैं।

किसी भी तरह की Redundancy विसंगतियाँ बढ़ाती है–क्योंकि Redundancy का मतलब है कि सूचना का कुछ हिस्सा दो जगह या दो बार represent किया (दर्शाया) गया है इसलिए यह संभावना बनी रहती है कि कभी अगर एक Data update हो जाए और दूसरा नहीं हो पाए।

अलग-अलग तरह की विसंगतियाँ नीचे दी गई हैं।

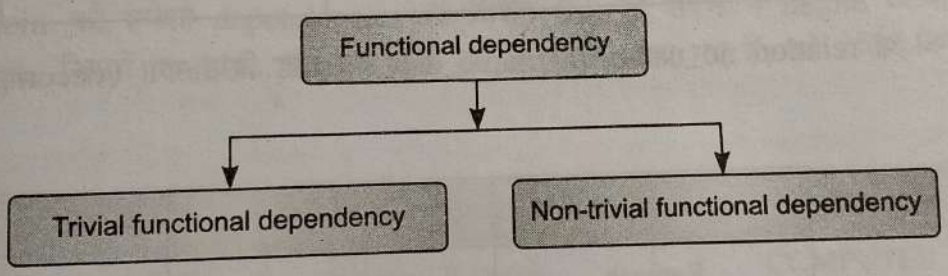
- **Insertion Anomaly (विसंगति):** अगर किसी refrencing relation में कोई tuple insert की जाए और refrencing attribute की value refence attribute में नहीं है, तो यह refrencing relation में 'insertion' की इजाजत नहीं देगा।
- **Deletion और Updation की विसंगति:** अगर किसी referenced relation की attribute value को कोई दूसरा relation (table) use (या reference) कर रहा हो तो उस tuple को delete नहीं किया जा सकता।
- **On Delete/Update Set Null:** अगर किसी Refrenced (जिसकी value use की जा रही हो किसी दूसरे relation के द्वारा) में से tuple को delete या update कर दिया जाए तो refrencing relation (जहाँ पर value इस्तेमाल हो रही हो इस value को NULL कर देगा।
- **On Delete/Update Cascade:** अगर कोई tuple refrenced relation में से delete या update की जाती है, तो यह refrencing relation में से भी उस row को delete या update कर देगा।

4.3 FUNCTIONAL DEPENDENCIES AND DECOMPOSITION

Functional dependency (कार्यात्मक निर्भरता) दो attribute के बीच के relationship को कहते हैं। आम तौर पर यह primary key और Non-key attribute के बीच (table के अन्दर) ही होती है। इसे नीचे दिए गए तरीके से दिखाते हैं

$$X \rightarrow Y$$

Left side (बायाँ भाग) को Determinant कहते हैं और right side (दाएँ भाग) को dependent कहते हैं।



1. Trivial Functional Dependency

- $A \rightarrow B$ यह एक Trivial Functional dependency है अगर B, A का Subset हैं।
- इस तरह की dependencies भी trivial होती हैं, जैसे $A \rightarrow A, B \rightarrow B$

2. Non-trivial Functional Dependency

- $A \rightarrow B$ एक Non-trivial functional dependency है अगर B, A का Subset नहीं है।
- जब A intersection B is NULL [A और B के बीच कोई चीज Common नहीं है] तो $A \rightarrow B$ एक Complete (पूरी तरह) Non-trivial dependency है।

Functional decomposition वह प्रक्रिया है जिसमें किसी organisation के functions (कार्यों) को आगे जाकर और detailed levels में तोड़ दिया जाता है (divide कर दिया जाता है)।

Decomposition के अन्दर, एक Function को पूरी detail के साथ समझाया जाता है, कुछ supporting functions के set के द्वारा किसी भी relation scheme R की Decomposition का मतलब है कि उसे दो या ज्यादा relation schema में बाँट दिया जाए, जिसमें प्रत्येक Schema के अन्दर R को कुछ attributes होंगे, यह R के

Attributes का subset होगा और सभी Schema मिलकर R के सभी attributes को represent (प्रदर्शित) करेंगे।

Decomposition गलत Design की कुछ Problems जैसे redundancy, inconsistencies (अनिरंतरता) और विसंगतियां आदि को दूर करती है।

Decomposition दो प्रकार की होती है।

1. Lossy Decomposition
2. Lossless Join Decomposition

Lossy Decomposition (किसी बदलाव/नुकसान के साथ Decomposition)

"किसी Relation R की R_1 और R_2 में decomposition lossy कहलाती है जब R_1 और R_2 के Join से वही relation ना मिले जो R का था" दो या ज्यादा relational schemes (या tables) को decompose करने का एक नुकसान यह है कि Original table को दुबारा प्राप्त करते वक्त कुछ information (सूचना) का loss हो जाता है।

नीचे दी गई table STUDENT को देखें जिसमें तीन attributes roll-no, Sname और department हैं–

Student:

Roll_no	*Sname*	*Dept*
111	Ramesh	COMPUTER
222	Shalu	ELECTRICAL

इस Relation को दो relation no_name और name_dept, में तोड़ दिया गया (decompose कर दिया)

Name_dept:

Sname	*Dept*
Ramesh	COMPUTER
Shalu	ELECTRICAL

No_name

Roll_no	*Sname*
111	Ramesh
222	Shalu

Lossy decomposition में अगर इन दो relations को दोबारा neutral join से जोड़ा जाए तो duplicate tuples बन जाती है–

stu_joined

Roll_no	*Sname*	*Dept*
111	Ramesh	COMPUTER
111	Ramesh	ELECTRICAL
222	Shalu	COMPUTER
222	Shalu	ELECTRICAL

यह decomposition एक गलत और Lossy (नुकसान देय) decomposition है।

Lossless Join Decomposition (बिना नुकसान के)

"किसी Relation R की R_1 और R_2 में decomposition lossless कहलाती है जब R_1 और R_2 को JOIN करने पर वही relation मिले जो R का था।"

कोई relational table दो या ज्यादा छोटी tables में इस प्रकार decompose की जाती है कि designer decomposed parts को JOIN करने पर original table को वैसी की वैसी प्राप्त कर सके। इसे Lossless_Join (या non-additive Join) decomposition कहते हैं।

इसे non-additive decomposition भी कहते हैं।

Lossless join को हमेशा dependencies के विशेष Set के संदर्भ में define किया जाता है। नीचे दी गई STUDENT table जिसमें Roll No. Sname और Department दिए गए हैं, को देखिए–

Student:

Roll_no	*Sname*	*Dept*
111	Ramesh	COMPUTER
222	Shalu	ELECTRICAL

इस relation को दो relations में बाँटा जा सकता है, Stu_name और Stu_debt.

Stu_name:

Roll_no	*Sname*
111	Ramesh
222	Shalu

Stu_dept:

Roll_no	*Dept*
111	COMPUTER
222	ELECTRICAL

इन relations को अगर एक जैसे column Roll no. से Join किया जाए तो जो relation मिलेगा वह Main relation जैसा ही होगा।

stu_joined:

Roll_no	*Sname*	*Dept*
111	Ramesh	COMPUTER
222	Shalu	ELECTRICAL

Lossless-join decomposition में कोई भी Duplicate tuples (rows) नहीं बनती

4.4 NORMALIZATION की प्रक्रिया

- Integrity constraints किसी अचानक होने वाले damage से database की सुरक्षा करते हैं।
- Integrity constraints database के Predicates (नियम या विधेय) की तरह काम करते हैं।
- **Domain constraints:**
  - ➢ Domain के लिए बनाए गए नियमों को Domain constraints कहा जाता है।
  - ➢ सामान्य Domain constraints attributes के Data type (attribute में किस प्रकार का data आएगा, numerical integer आदि) से Define करके बनाए जाते हैं, Data types Integer, Double float इत्यादि होते हैं।
  - ➢ हम Domain को Create domain से परिभाषित (define) कर सकते हैं और उसके साथ constraint भी लगा सकते हैं जैसे कि–

create domain hourly-wage **numeric**(5,2)

constraint wage-value-test check(value >= 4,00)

यहाँ हमनें hourly wage की domain define कर दी और यह constraint भी लगा दिए की वह numeric value हो सकती है और हमेशा 4.00 या 4.00 से बड़ी होगी।

➢ Domain को Define करने के और भी उदाहरण हैं जैसे–

create domain account-number **char**(10)

constraint account-number-null-test check(value not null)

create domain account-type **char**(10)

constraint account-type-test

check(value in ("Checking", "Saving"))

अब DBMS अपने आप attribute account_type के लिए केवल दो ही values डालने देगा–

Checking या Saving

- **Referential Integrity:**
  - ➢ **Foreign key:** यदि दो table R और S आपस में संबंधित है, और K_1 और K_2 relation की दो Primary key है, और K_1 table S का एक attribute भी है, और हम यह चाहते कि table 'S' की प्रत्येक Row के संदर्भ में 'R' में भी कोई Row होनी चाहिए, तो हम K_1 को Foreign key बना देते हैं, उदाहरण के लिए मान लीजिए। LIBRARY का एक database है जिसमें table है 'BORROWED BY' इस table में दो field हैं Card. no. और Book No. [मतलब किस कार्ड न. ने कौन सी Book Borrow की है।

    अब BORROWED BY TABLE की प्रत्येक ROW USER TABLE की किसी न किसी table से संबंधित होगी जिनका Card no. same होगा और Borrowed by table की प्रत्येक Row,'Book' table की किसी न किसी Row से संबंधित होगी क्योंकि Book table में सभी Book Nos. होंगे और Borrowed by table का हर Book No. उन्हीं में से होगा, तो हम Card no. और Book No. को Borrowed by relation में Foreign keys कह सकते हैं।
  - ➢ या हम कह सकते हैं कि Borrowed by table की प्रत्येक Row, Book और User table की किसी न किसी Row से संबंधित है।
  - ➢ इस तरह की Refrential requirements (जरूरत) को ही Refrential Integrity कहते हैं।
  - ➢ Refrential Integrity constraints को परिभाषित करने का मतलब है, कि किसी table में कुछ attributes को foreign key की तरह define (परिभाषित) करना जो किसी अन्य table/tables में primary key हैं।
- **Functional Dependencies:**
  - ➢ मान लीजिए किसी Relation की Schema R है, और $\alpha \subseteq R$ और $\beta \subseteq R$ तो R पर एक Functional dependency $\alpha \rightarrow \beta$ काम करती है, अगर, किसी भी table में जिसकी Schema R है, किसी दो row r_1 और r_2 के लिए attribute α की value same होगी तो attribute β की value भी same होगी।
  - ➢ उदाहरण के लिए नीचे table को देखिए–

Seq	*A*	*B*	*C*	*D*
1	a_1	b_1	c_1	d_1
2	a_1	b_2	c_1	d_2
3	a_2	b_2	c_2	d_2
4	a_2	b_3	c_2	d_3
5	a_3	b_3	c_2	d_4

अब हमें देखना है कि $A \rightarrow C$ लागू होती है या नहीं ऐसे जोड़े (pairs) देखिए जहाँ A की Value same है।

- ♦ Row No. 1 और 2 में A के मान (value) Same है और C की Value भी Same है।
- ♦ Row no. 3 और 4 में, A की Value same है और C की भी Same है।
- ♦ A की और किसी row में same value नहीं है अत: $A \rightarrow C$ लगू होती है।

अब हमें देखना है कि $C \rightarrow A$ लागू होती है या नहीं–

- Row 1 और 2 में से C की value same है और A की भी same है।
- Row 3 और 4 में C की value same है और A की भी Same है।
- Row 4 और 5 में C की Value same है **परन्तु** 4 और 5 में A की Value same नहीं है, अतः $C \rightarrow A$ **लागू नहीं होती।**

हम यह भी सिद्ध कर सकते हैं कि $AB \rightarrow D$ लागू होती है इसके लिए दो ऐसी row ढूँढ़ें जहाँ A और B दोनों की Same value हो।

- कोई row ऐसी नहीं है जहाँ A और B की Same value हो इसलिए $AB \rightarrow D$ लागू होते हैं।

➢ अगर K किसी Relation R की Super key तो इसका मतलब है, कि Functional dependency $K \rightarrow R$ लागू होती है, और इसका उल्टा भी सही है मतलब कि $R \rightarrow K$ भी लागू होती है।

➢ **Armstrong के नियम:** मान लीजिए कोई Relation R दिया हुआ है और उस पर Functional dependencies F लागू होती है तो इन नियमों से बाकी सभी functional dependencies निकाली जा सकती है, जो दिए गए relation R और Functional dependencies F से निकलती है।

- **Reflexivity rule:** अगर किन्हीं attributes के set में $\beta \subseteq \alpha$ तो $\alpha \rightarrow \beta$ लागू होती है।
- **Augmentation rule:** अगर $\alpha \rightarrow \beta$ किसी attributes के set के लिए लागू होती है तो $\gamma\alpha \rightarrow \gamma\beta$ भी लागू होगी।
- **Transitivity rule:** अगर $\alpha \rightarrow \beta$ लागू होती है और $\beta \rightarrow \gamma$ लागू होती है तो $\alpha \rightarrow \gamma$ भी लागू होगी।

➢ Functional dependencies को निकालने के लिए कुछ नए नियम भी बनाए गए क्योंकि Armstrong के नियम लंबे और थकाने वाले हैं। जबकि यह भी सही है कि हम सिर्फ Armstrong Rules' से सभी Functional dependencies निकाल सकते हैं।

- **Union rule:** अगर $\alpha \rightarrow \beta$ लागू होती है और $\alpha \rightarrow \gamma$ भी लागू होती है तो, $\alpha \rightarrow \beta\gamma$ भी लागू होगी।
- **Decomposition rule:** अगर $\alpha \rightarrow \beta\gamma$ लागू होती है तो $\alpha \rightarrow \beta$ और $\alpha \rightarrow \gamma$ भी लागू होगी।
- **Pseudotransitivity rule:** अगर $\alpha \rightarrow \beta$ लागू होती है और $\gamma \rightarrow \beta\delta$ भी लागू होती है तो $\alpha\gamma \rightarrow \delta$ भी लागू होगी।

➢ **Functional dependencies का closure (पूरा सेट):** मान लीजिए कि किसी relation schema R के लिए Functional depenedencies का set F दिया गया है। तो ऊपर दिए गए सारे नियमों से हम और नई functional dependencies निकाल सकते हैं। इन सभी नई functional dependencies और दी गई functional dependencies F के पूरे Set (समूह) को functional dependencies का Closure कहा जाता है और इसे F से denote किया जाता है।

➢ मान लीजिए कोई Schema R है, R = (A, B, C, G, H, I) और functional dependencies का set F है जिसमें नीचे दी गई functional dependencies शामिल है:

- $A \rightarrow B$ 1
- $A \rightarrow C$ 2
- $CG \rightarrow H$ 3
- $CG \rightarrow I$ 4
- $B \rightarrow H$ 5

- अब ऊपर दिए गए नियमों से जितनी और functional dependencies निकल सकती हैं उतनी निकाल लीजिए जैसे कि
- 1 और 5 पर transitivity rule लगाकर कम नई functional dependencies $A \rightarrow H$ निकाल सकते हैं।
- $CG \rightarrow HI$ को हम functional dependencies 3 और 4 पर UNION rule लगाकर निकाल सकते हैं।
- $AG \rightarrow I$ को हम 2 और 3 पर Pseudo transitivity लगाकर निकाल सकते हैं।

Normal Forms

- किसी Database का अगर design खराब हो जाए तो उसमें निम्नलिखित कमियाँ हो सकती हैं।
  - सूचना का दोंहरीकरण (Repetition)
  - किसी भी सूचना को बताने में असमर्थ होना।
  - Data की Integrity बनाए रखने में नाकाम होना।
- किसी database की theory को normal forms वह मानदंड (criteria) प्रदान करते हैं जो यह बताता है कि कोई table विसंगतियों (errors) के प्रति कितना संवेदनशील है या भेद्य है।
- जितना ज्यादा Normal form table में applicable होगा उतना ही कम वह table विसंगतियों या असंगतियों (Inconsistencies) के प्रति संवेदनशील या भेद्य होगी।
- प्रत्येक table का "Highest normal form" (HNF) होता है–परिभाषा के अनुसार कोई table हमेशा अपने HNF और उसके नीचे के सभी Normal forms की जरूरतों को पूरा करती है, और परिभाषा से ही, कोई table अपने HNF से higher normal forms की जरूरतों को पूरा नहीं करती।
- Normal forms का सामान्य अनुक्रम (order) यह है जैसे कि First Normal form (1 NF), Second Normal Form (2NF), Third Normal Form (3 NF), Fourth Normal Form (4 NF), Fift Normal Form (5 NF)।
- हम ऊपर दिए गए अनुक्रम में से सिर्फ 3 NF तक discuss करेंगे और इसके अलावा एक और Normal form [BCNF] के बारे में पढेंगे।

First Normal Form:

- 'Date' की 1 NF की परिभाषा के अनुसार, कोई Table 1 NF में है अगर वह किसी Relation से Isomorphic" (समरूप या Same structure) है। मतलब की यह नीचे दी गई पाँच शर्तें पूरी करती है:
  1. Row का कोई भी ऊपर से नीचे क्रम नहीं होता।
  2. इसी तरह Column का भी बाएँ से दाएँ क्रम नहीं होता।
  3. कोई भी Duplicate Row नहीं होती।
  4. जहाँ कोई Row और Column मिलते हैं वहाँ केवल एक Value होनी चाहिए। (जो Domain में आती हो।) और कुछ नहीं।
  5. सभी column regular होने चाहिए [मतलब की rows के अन्दर कोई छुपे हुए Components नहीं होने चाहिए, जैसे कि Row ID's, object Ids, या hidden timestamps]
- जो Table (या view) INF की परिभाषा को पूरा नहीं करती उनके उदाहरण है:
  - वह table जिनमें कोई unique key ना हो। इस तरह की tables में duplicate rows आ जाती है और वह ऊपर दी गई शर्तों में से 3 नं. शर्त को पूरा नहीं करती।

- वह view जिनकी जरूरत होती है कि Result किसी क्रम (Particular Order) में ही हो। यह शर्त न. एक को पूरा नहीं करती। किसी True relation में rows एक दूसरे के हिसाब से क्रम में नहीं होती।
- ऐसी table जिसमें कम से कम एक Null attribute हो। यह शर्त न. 4 का उल्लंघन है, जो कहती है कि प्रत्येक Field की अपने Column की Domain के हिसाब से कोई न कोई value होनी चाहिए। यह भी ध्यान रखने योग्य है कि Condition 4 में कुछ मतभेद हैं। बाद में Codds के relation model में Null के लिए कुछ स्पष्ट प्रावधान (Explicit Provisionals) किए गए।
- Codd के अनुसार, "Domin की Values जिसके ऊपर relation को define किया जाता है।" वह DBMS के हिसाब से atomic होनी चाहिए। Codd की परिभाषा के अनुसार atomic value वह है जो आगे और छोटे टुकड़ों में नहीं बाँटी (या तोड़ी) जा सके। मतलब कि किसी Field के इस तरह टुकड़े नहीं होने चाहिए कि उसमें एक से ज्यादा तरह के Data इस तरह से हो कि एक Part का DBMS के लिए क्या मतलब है यह उसी Field के दूसरे Part पर निर्भर हो।
- मान लीजिए की कोई Designer Customers के नाम और Telephone नं. record करना चाहता है तो वह एक Customer table बनाए जो ऐसे दिखती है–

Customer:

Customer ID	*First Name*	*Surname*	*Telephone Number*
123	Akash	Bajpai	555-861-2025
456	Vikas	Bajpai	555-403-1659
789	Abhas	Bajpai	555-808-9633

- तभी designer को यह जरूरत महसूस होती है कि कुछ customers के एक से ज्यादा telephone no. record करने पड़ सकते हैं। तो उसके अनुसार यह करने का सबसे आसान तरीका यह है कि "Telephone No." field एक से ज्यादा value ले सके।

Customers ID	*First Name*	*Surname*	*Telephone Number*
123	Akash	Bajpai	555-861-2025
456	Vikas	Bajpai	555-403-1659 555-776-4100
789	Abhas	Bajpai	555-808-9633

यह मानते हुए कि Telephone number column को किसी Domain में (जैसे की String को Domain जिसकी Length 12 Characters की हो) में Define किया गया है, ऊपर दिया हुआ चित्रण INF में नहीं है। क्योंकि INF (या कह सकते हैं की RDBMS) एक ही field में Column की Domain में से value को एक ज्यादा बार नहीं डालने देता।

- **Repeating group across columns:** हो सकता है कि Designer इस दिक्कत से निपटने के लिए Telephone No. के लिए अलग-अलग Columns बना दे–

Customer ID	*First Name*	*Surname*	*Tel. No. 1*	*Tel. No. 2*	*Tel. No. 3*
123	Akash	Bajpai	555-861-2025		
456	Vikas	Bajpai	555-403-1659	555-776-4100	555-403-1659
789	Abhas	Bajpai	555-808-9633		

यह representation क्योंकि Null Column का Use करता है, यह INF के लिए Date की परिभाषा को Confirm नहीं करता हालांकि View को इस तरह से लिया गया है कि Null Columns भी allowed है परन्तु Design INF के हिसाब से नहीं है। Tel. No. 1, Tel. No. 2 और Tel. No. 3 एक ही Domain से है, और उनका मतलब एक ही है। Telephone No. को 3 heading में बाँटने से logical problem हो सकती है जैसे–

- Table से Query बनाने में दिक्कत 1 इस तरह के प्रश्नों के उत्तर देना कि "Telephone No. 1 किस customer का है?" या Customers का कौन सा जोड़ा यह telephone no. use करता है? कभी-कभी मुश्किल हो जाता है।
- Customer से telephone no. का एक unique (अद्वितीय) link नहीं बनता। Customer 789 को गलती से कभी Tel. No. 2 भी दिया जा सकता है जो Tel No. 1 के बराबर हो।
- किसी भी Customer के केवल तीन Telephone No. ही enter किए जा सकते हैं। अगर किसी customer के पास 4 telephone no. हों तो हम सिर्फ 3 का record रखेंगे और एक को बिना record किए छोड़ देंगे। इसका मतलब है कि Database का Design Business की process को ही restrict (बाधित) कर रहा है।
- **Repeating groups within column (Column को अन्दर ही दो से ज्यादा values):** अब designer हो सकता है कि Telephone नं. का पहले की तरह एक ही column कर दे, और उस column की Domain को ही change (बदल) कर दे, उसकी String length को अनेक Mobile No. के लिए पर्याप्त बना कर।

Customer ID	*First Name*	*Surname*	*Telephone Numbers*
123	Akash	Bajpai	555-861-2025
456	Vitas	Bajpai	555-03-1659, 555-776-4100
789	Abhas	Bajpai	555-808-9633

यह Design Date की परिभाषा के हिसाब से तो INF के साथ मेल खाता है परन्तु Codd की परिभाषा से नहीं। इसमें बहुत से Design के मुद्दे हैं। 'Telephone No. शीर्षक' शब्दों के हिसाब से उलझा हुआ है। क्योंकि अब यह Telephone numbers के बारे में बता रहा है या Telephone numbers की list के बारे में बता रहा है या कुछ और बता रहा है। इस तरह के सवाल की "Customers के कौन से Pairs कौन सा Telephone No. share कर रहे हैं?" इनके जवाब देना कठिन है। अब कुछ ऐसा चाहिए जो अकेले Telephone no. के साथ-साथ Telephone No. की List की जरूरतों को भी पूरा कर सके। इस तरह के Design के साथ RDBMS में Telephone numbers पर अथपूर्ण constraints लगाना भी मुश्किल भरा काम है।

➢ ऐसा Design जो 1 NF के साथ Comply (अनुपालन) करे: कोई Design जो स्पष्ट रूप से 1 NF में है, वह यहाँ दो Tables का इस्तेमाल करेगा: एक Customer Name table आरै दूसरी Customer telephone number table

Customer Name:

Customer ID	*First Name*	*Surname*
123	Akash	Bajpai
456	Vikas	Bajpai
789	Abhas	Bajpai

Customer Telephone:

Customer ID	*Telephone Number*
123	555-861-2025
456	555-403-1659
456	555-776-4100
789	555-808-9633

इस तरह के Design में Telephone Nos. के repeated groups नहीं आ रहे, और प्रत्येक Customer और telephone no. का अपने record में ही आ रहा है।

यहाँ यह ध्यान देने योग्य बात है कि यह Design second and third normal form (2 NF and 3 NF) की जरूरतों को भी पूरा करता है।

Second Normal Form:

- 2 NF को मूल रूप से EF Codd ने 1971 में Design किया था। कोई 1 NF table 2 NF होगी केवल और केवल अगर, उसमें कोई Candidate Key K दी हुई है और कोई attribute A है जो Candidate key का हिस्सा नहीं है, और A पूरी Candidate Key K पर निभर करेगा ना कि उसके किसी हिस्से पर।
- अगर उसके Non-prime attributes Functionally प्रत्येक पूरी Candidate key पर निर्भर करते हों। Non-prime attributes वह होते हैं जो किसी Candidate key के अंतर्गत नहीं आते।
- ध्यान दें, जब भी किसी 1NF table में कोई composite candidate key (वह candidate key जिनमें एक से ज्यादा attributes हों) ना हो तो वह table अपने आप ही 2NF में होती है।
- नीचे दी गई table देखिए जिसमें employee's की skill के बारे में बताया गया है–

Employees' skills: यहाँ पर ना तो (Employee) और ना ही (skill) Candidate keys हैं क्योंकि एक employee की बहुत सारी skill हो सकती हैं और किसी एक skill को बहुत सारे Employees रखते हों तो इस table के लिए सिर्फ composite key [Employee_skill], ही Candidate key का काम कर सकती है।

Employee	*Skill*	*Current Work Location*
Akash	Typing	114 Main Street
Akash	Shorthand	114 Main Street
Akash	Whittling	114 Main Street
Brajesh	Light Cleaning	73 Industrial Way
Sanjay	Alchemy	73 Industrial Way
Sanjay	Flying	73 Industrial Way
Santosh	Light Cleaning	73 Industrial Way

बचा हुआ attribute, current work location candidate key [employee, skill] के केवल एक part employee पर निर्भर करता है। इसलिए यह table 2NF में नहीं है। यहाँ ध्यान दीजिए की current work location field में किस तरह Redundancy (values में Duplicate) हुआ है: यहाँ तीन बार वही चीज बताई गई है कि Aakash 114, main street में काम करता है। दो बार यह बताया गया है कि Sanjay 73 Industrial way में काम करता है। यह Duplicasy table को Update करते समय विसंगतियों की चपेट में ला सकती है जैसे, यह हो सकता है कि Aakash की location को उसके 'typing' और 'Shorthand' के records में तो update कर दिया जाए पर उसके whittling record में location update ना हो पाए, तो अगर यह प्रश्न पूछा जाए कि 'Aakash की Current work Location क्या है? तो data विरोधाभास वाला उत्तर देगा।

- इस Design के रूप में 2 NF इस सूचना को दो tables एक Employee table जिसकी candidate key employee होगी, और दूसरी table 'employee skill' जिसकी candidate key [Employee, skill] होगी, में बाँट देगा।

Employees:

Employee	*Current Work Location*
Akash	114 Main Street
Brajesh	73 Industrial Way
Sanjay	73 Industrial Way
Santosh	73 Industrial Way

Employees' Skills:

Employee	*Skill*
Akash	Typing
Akash	Shorthand
Akash	Whittling
Brajesh	Light Cleaning
Sanjay	Alchemy
Sanjay	Flying
Santosh	Light Cleaning

इनमें से कोई भी Table अब update करने पर विसंगतियों से प्रभावित नहीं होगी।

- जरूरी नहीं है कि सभी 2NF table update की विसंगतियों से free होती है, नीचे उदाहरण में एक table दी गई है, जिसे Update करने पर विसंगतियाँ (अलग-अलग Data) पैदा हो सकती है।

Tournament Winners:

Tournament	*Year*	*Winner*	*Winner Date of Birth*
Hockey	1998	India	14 March 1977
Cricket	1998	West Indies	21 July 1975
Football	1999	USA	28 September 1968
Hockey	1999	West Indies	21 July 1975
Cricket	1999	India	14 March 1977

जबकि Winner और winner date of birth whole key [Trournament/Year] से निकल सकते हैं ना कि उसके किसी Part से तब भी किसी विशेष Winner/Winner date of birth के combination में Duplicate या Multiple records हो सकते हैं। इससे update की विसंगतियाँ उत्पन्न होंगी अगर update को लगातार ध्यान से नहीं किया गया तो हो सकता है किसी एक winner की दो अलग-अलग Date of birth दिखने लग जाए। इस प्रकार की Problem एक Transitive dependency है जो जिसके Subject winner Date of birth attribute है। Winner date of birth असल में winner पर निर्भर करता है।

- यह Problem third normal form (3NF) से दूर हो सकती है।
- ध्यान दें–Primary key, के अलावा table में और candidate keys हो सकती है; तो यहाँ यह स्थापित करना जरूरी है कि कोई भी non-prime attribute (दिखाना) ऐसा नहीं है जो candidate key के किसी एक part पर निर्भर करता हो।

Third Normal Form:

- 3NF को E.F. Codd के द्वारा 1971 में परिभाषित किया गया था कोई table 3 NF में होगी केवल और केवल नीचे दी गई दोनों शर्तें लागू होगी–
  - Relation R (table) Second normal form में अवश्य होनी चाहिए।
  - Table R का प्रत्येक Non-prime attribute R की Candidate key पर directly निर्भर होना चाहिए।
  - **ध्यान दें:**
    - **R का Non-prime attribute:** वह attribute है जो R की किसी Candidate key से belong नहीं करता (संबंधित नहीं है)
- 3 NF की एक और परिभाषा जो Codd's की परिभाषा के बराबर है, Carlo Zonilio ने 1982 में दी थी। जो कहती थी कि कोई table 3NF होगी अगर प्रत्येक Functional dependency $X \rightarrow A$ के लिए नीचे दी गई शर्तों में से कम से कम एक शर्त लागू होती हो:
  - X के अन्दर ही A शामिल हो ($X \rightarrow A$ एक trivial functional dependency हो) या
  - X एक super key हो, या
  - $X \rightarrow A$ का प्रत्येक attribute एक prime attribute हो (मतलब की candidate key में शामिल हो)
- Zonilio की परिभाषा 3 NF और Buyce-codd Normal form (BCNF) के बीच के अंतर को स्पष्ट रूप से समझाती हैं। BCNF सिर्फ तीसरे वाले alternative को हटा देता है। (कि $X \rightarrow A$ का प्रत्येक attribute prime होना चाहिए।)

- 2NF और 3NF के बीच का अंतर यह है कि: अगर Non-key attributes पूरी (whole key) पर निर्भर करता है (ना कि उसके किसी Part पर) तो यह सुनिश्चित करता है कि table 2nF में है और अगर non-key attribute और किसी पर नहीं केवल key पर ही निर्भर करता है तो table 3NF में है।
- उदाहरण के लिए ऊपर दी गई table को ही लेते हैं।

Tournament Winners:

Tournament	*Year*	*Winner*	*Winner Date of Birth*
Hockey	1998	India	14 March 1977
Cricket	1998	West Indies	21 July 1975
Football	1999	USA	28 September 1968
Hockey	1999	West Indies	21 July 1975
Cricket	1999	India	14 March 1977

यह table 2 NF में है परन्तु 3 NF में नही है। 3 NF की शर्त पूरी नहीं होती क्योंकि Non-prime attribute winner date or birth candidate key (tournament/year) पर directly निर्भर नहीं करता है यह candidate key पर non-prime attribute winner के जरिए निर्भर करता है। सच्चाई यह है कि क्योंकि 'winner date of birth' Functionally winner पर निर्भर करता है यह table को logical असंगतियों की चपेट में ला सकता है क्योंकि इससे एक ही व्यक्ति की अलग-अलग Records में अलग--अलग date of birth हो सकती है। सभी तथ्यों को बिना 3NF की शर्तों को तोड़े बिना दिखाने के लिए इस table को दो tables में दिखाना जरूरी है–

Tournament Winners:

Tournament	*Year*	*Winner*
Hockey	1998	India
Cricket	1998	Pak
Football	1999	USA
Hockey	1999	USA
Cricket	1999	India

Player Dates of Birth:

Player	*Date of Birth*
India	14 March 1977
Pak	21 July 1975
USA	28 September 1968

Boyce-Codd Normal Form [BCNF]

- यह third Normal form [3NF] का थोड़ा बढ़िया संस्करण (version) है। कोई table BCNF में है सिर्फ अगर, उसकी प्रत्येक non-trivial (dependencies) $X \rightarrow Y$ में, X एक super key हैं। मतलब X या तो candidate key है या एक उसका super set है।
- ध्यान से देखे ऊपर दिए गए table 'Tournament Winners' और 'player date of birth' का set 3NF में है और साथ ही BCNF में भी हैं
- कोई दुर्लभ case ही होगा जहाँ कोई 3 NF table, BCNF की जरूरतों को पूरा नहीं करती। एक 3NF table जिसमें विभिन्न overlapping candidate key नहीं होती, वह guranteed एक BCNF table होती है।
- एक 3NF table का उदाहरण जो BCNF नहीं, है:

Today's Court Bookings:

Court	*Start Time*	*End Time*	*Rate Type*
1	09:30	10:30	SAVER
1	11:00	12:00	SAVER
1	14:00	15:30	STANDARD
2	10:00	11:30	PREMIUM-B
2	11:30	13:30	PREMIUM-B
2	15:00	16:30	PREMIUM-A

यहाँ दो court उपलब्ध है और चार तरह के Rate है।

- Saver Court 1 में members के लिए
- STANDARD Court 1 में non-members के लिए
- PREMIUM-A Court 2 में Members के लिए
- PREMIUM-B Court 2 में non members के लिए तो Rate type → court, non-trivial functional dependency है, जो यहाँ लागू होती है।

इस table की candidate keys है–

- {Court, Start Time)
- {Court, End Time}
- {Rate Type, Start-Time}
- {Rate Type, End Time}

- इस table में कोई non-prime attribute नहीं है, मतलब सभी attributes candidate key की belong करते है। इसलिए table 2NF और 3NF दोनों के नियमों का पालन करती है।
- परन्तु table BCNF का पालन नहीं करती क्योंकि dependency: Rate type→Court जो attribute (rate type) को निर्धारित करती है, वह एक super key नहीं है।
- इस Design को नीचे दिए गए तरीके से बदला जा सकता है ताकि वह BCNF पर खरा उतरे:

Rate Types:

Rate Type	Court	Member Flag
SAVER	1	Yes
STANDARD	1	No
PREMIUM-A	2	Yes
PREMIUM-B	2	No

Today's Bookings:

Rate Type	Start Time	End Time
SAVER	09:30	10:30
SAVER	11:00	12:00
STANDARD	14:00	15:30
PREMIUM-B	10:00	11:30
PREMIUM-B	11:30	13:30
PREMIUM-A	15:00	16:30

यहाँ Rate type table के लिए candidate key है [Rate type] और [Court, Member flag]; और todays booking table के लिए candidate key है– [Rate type, start time] और [Rate type, end time] दोनों tables BCNF में है।

नीचे दी गई Table देखिए–

Lending:

branch-name	*branch-city*	*assets*	*customer-name*	*loan-number*	*amount*
Sadar	Agra	200000	Ram	L-12	12000
Sanjay-place	Agra	100000	Ram	L-13	13000

यह table loan की सूचना store करती है। इस table में निम्नलिखित problem है–

- ♦ क्योंकि एक branch में बहुत सारे loan होते हैं। और प्रत्येक loan जो एक branch से लिया गया है उसके लिए एक row होगी। उस row में branch-name, branch-city और assets के column की value एक जैसी होगी, और यह data का repetition होगा।
- ♦ किसी एक branch के लिए branch_city या assets को update करना हो तो हमें table की प्रत्येक row को update करना पड़ेगा यह प्रक्रिया बहुत costly होगी।
- ♦ और अगर हमने किसी एक row को बिना update के छोड़ दिया तो, branch city या assets के लिए एक से ज्यादा value हो सकती है जो Data की संप्रभुता (Integrity) उल्लंघन है।
- ♦ अगर किसी branch में कोई loan नहीं है, तो इस table में कोई entry ही नहीं होगी और bank की branches की पूरी सूचना database से प्राप्त नहीं होगी।

Decomposition:

- यह सभी दिक्कतें इस table को decompose करके (अलग-अलग table में बाँट कर) दूर हो सकती है। अगर R कोई relation है तो R_1, R_2, R_3 ... R_n R के decomposition का set होगा अगर $R = R_1 U R_2 U R_3 U ... U R_n$. यह ध्यान रखना चाहिए कि इस set को किसी भी pair (जोड़ा) R_i और R_{i+1} में कम से कम एक common attribute होगा जिससे कि उन्हें Join operation से दोबारा जोड़ा जा सके।
- परन्तु इस table के सभी decomposition error-free नहीं होंगे।
- उदाहरण के लिए अगर हम अपनी lending table में से दो नई table बना लें जैसे–

Branch-customer-schema = (branch-name, branch-city, assets, customer name)

Customer-loan-schema = (customer-name, loan-number, amount)

जो दो tables बनेंगी वो इस तरह होंगी।

Branch-Customer:

branch-name	*branch-city*	*assets*	*customer-name*
Sadar	Agra	200000	Ram
Sanjay-place	Agra	100000	Ram

Customer-Loan:

customer-name	*loan-number*	*amount*
Ram	L-12	12000
Ram	L-13	13000

अब Loan No. L-12 की branch पता करने के लिए अगर हम दोनों tables को join करें तो हमे नीचे वाली table मिलेगी

Branch-customer ⋈ Customer-loan =

branch -name	*branch-city*	*assets*	*customer-name*	*loan-number*	*amount*
Sadar	Agra	200000	Ram	L-12	12000
Sadar	Agra	200000	Ram	L-13	13000
Sanjay-place	Agra	100000	Ram	L-12	12000
Sanjav-place	Agra	100000	Ram	L-13	13000

इस Join के अनुसार दोनों loan दोनों Branches से लिए गए हैं। यह सूचना की हानि (Information loss) और wrong information का एक उदाहरण है। ऐसा इसलिए हुआ कि decomposition करते वक्त दो tables में जो common column (attribute) रखना था उसे चुनने में हमने गलती कर दी।

- **Loss-less Join Decomposition:** (इस तरह decomposed करना कि Join करने पर कोई loss न हो) एक decomposition (R1, R2 ... Rn) किसी relation schema R की Lossless Join Decomposition होगी अगर Schema R के सभी उचित relations r के लिए:

$$r = \Pi_{R_1}(r) \bowtie \Pi_{R_1}(r) \bowtie \ldots \bowtie \Pi_{R_n}(r)$$

या दूसरे शब्दों में Decomposition के बाद सभी decomposed (अलग-अलग) tables को Join करने के बाद जो table मिलेगी उसमें वही data होना चाहिए जो decomposition से पहले की original table में था।

- वरना उसे Lossy-Join decomposition कहेंगे।

Dependency Preservation: यह decomposition की एक और जरूरी property है। मान लो की दी हुई functional dependencies का एक Set F है, जो किसी भी relation पर जो Schema R पर आधारित है, पर लागू होती हैं। तो उन सभी Functional dependencies का set जो Subschema R_1 पर लागू होती है को F1 relation कहते हैं, जिसमें F की वह सभी Functional dependencies शामिल है, जिनमें सिर्फ R_1 के attributes शामिल हैं। तो अगर R की decposition $[R_1, R_2 \ldots R_n$, इस तरह हो की संबंधित Functional depenedencies जो उन पर लागू होती है वह हो [F1, F2 ... Fn, हो तो निम्नलिखित सच होना चाहिए:

$$F^+ = \{F_1 \cup F_2 \cup \ldots \cup F_n\}^+.$$

इस तरह की decomposition को dependency Preserving कहते हैं।

उदाहरण के लिए:

मान लीजिए Schema R = {A, B, C, D} इस तरह है कि नीचे दी गई Functional dependency उस पर

लागू होती है–

$$F = \{A \rightarrow B, A \rightarrow BC, C \rightarrow D\}.$$

अब R को दो tables में decompose कर दिया गया है $R_1 = [A, B]$, $R_2 = [B, C, D]$, तो R_1 पर जो Functional dependencies लागू होगी वह होगी $F_1 = \{A \rightarrow B\}$ [नोट: F_1 में F की वे सभी Functional dependencies होनी चाहिए जिनमें सिर्फ R_1 के attributes ही हो, और R_2 पर जो Functional dependencies लागू होगी $F_2 = \{C \rightarrow D\}$/ अब अगर हम $F_1 \cup F_2$ करते हैं तो हमें जो Functional Dependencies मिलेगी वह है $\{A \rightarrow B, C \rightarrow D\}$ जिसमें F की एक Functional Dependencies $A \rightarrow BC$ वही है, तो यह एक Dependency preservation decomposition नहीं है क्योंकि ये सभी Functional dependencies को सुरक्षित नहीं रखता। अगर हम R को इस तरह Decompose करते हैं कि $R_1 = \{A, B, C\}$ और $R_2 = \{C, D\}$ तो $F_1 = \{A \rightarrow B, A \rightarrow BC\}$ और $F_2 = \{C \rightarrow D\}$ तो $F_1 \cup F_2$ होगा $\{A \rightarrow B, A \rightarrow BC, C \rightarrow D\}$.

यह decomposition dependencies preserving है।

Functional Dependencies के इस्तेमाल से Normalisation करना

- Lossless-Join Decomposition FD की इस्तेमाल करके:
  - माना की R एक relation schema है और F, R पर लागू होने वाली Functional dependencies का Set है। माना की R_1 और R_2, R के दो decomposition है, यह decomposition lossless join decomposition होगा, अगर नीचे दी हुई functional dependencies में से कोई एक F^+ में शामिल होगी–
    - $R_1 \cap R_2 \rightarrow R_1$
    - $R_1 \cap R_2 \rightarrow R_2$
  - **उदाहरण:** ऊपर इस्तेमाल किए गए lending schema (table) में

    Lending Schema = (branch-name, branch-city, assets, customer name, loan number, amount)

    इसमें जो FD लागू होती है वह है:

    branch-name → assets branch-city

    loan-number → amount branch-name

    तो इसको दो Schema में decompose करने पर:

    Branch-schema = (branch-name, branch-city, assets)

    Loan-info-schema = (branch-name, customer-name, loan-number, amount)

    यह एक losssless Join decomposition होगा क्योंकि

    Branch-schema ∩ Loan-info-schema — branch-name

    और हमारे पास एक FD *branch-name → assets branch-city*

    जिस पर augmentation rule लागू करने पर यह FD *branch-name → branch-name assets branch-city* के बराबर होगी, और *branch-name → Branch-schema.*

- Third normal form, FD का इस्तेमाल करने पर:
  - माना कोई relation R है जिस पर F Functional dependencies का minimal set है तब निम्नलिखित प्रक्रिया करें:

1. शुरूआत में relations का एक खाली set लें।
2. F की प्रत्येक FD के लिए, $\alpha \rightarrow \beta$ i = 1
   - एक relation add करो $R_1 = (\alpha, \beta)$ अगर और किसी relation में α, β नहीं है, i को एक से बढाएँ।
3. इस तरह के सभी relation add करने के बाद एक और R_1 = (R की कोई भी Candidates key) का Add करना पड़ेगा, अगर किसी और relation में candidate key नहीं है।

➢ Boyce_Codd Normal Form [BCNF], FD का इस्तेमाल करके:

1. माना R कोई relation है जो BCNF में नहीं है।
2. और माना $\alpha \rightarrow \beta$ FD है जो relation पर लागू होती है, परन्तु $\alpha \rightarrow R_1$ लागू नहीं होती (क्योंकि α R_1 की super key नहीं है)
3. Relation R_1 को दो relations से बदल दीजिए।
4. अब दोबारा सभी relation उन सभी FD's (जो उन पर लागू होती है) के साथ चेक करें और Step No. 1 पर वापस जाइए।

➢ **उदाहरण:**

- माना lending Schema = (branch_name, branch_city, assets, customer_Name, loan_number, amount) और Schema पर जो FD लागू होगी वो है:
  1. branch-name → assets branch-city
  2. loan-number → amount branch-name
- हम यह देख सकते हैं कि lending schema BCNF में नहीं है, और हम यह भी देख सकते हैं कि Functional dependencies branch_name → asset branch_city में. branch_name lending Schema की तरह super key नहीं है। अत: relation BCNF में नहीं है। तो कुछ नए rleation है: नीचे दिए गए Set है:
  1. Branch-schema = (branch-name, branch-city, assets)
     branch-name → assets branch-city
  2. Loan-info-schema = (branch-name, customer-name, loan-number, amount)
     loan-number → amount branch-name
- दोबारा relations के नए Set में हम यह देख सकते हैं कि Loan-info-schema BCNF में नहीं है क्योंकि Loan Number super key नहीं है। इसलिए हम इसे दोबारा decompose करेंगे और जो relation के set बनेंगे वे होंगे:

  Branch-schema=(branch-name, branch-city, assets)
  branch-name → assets branch-city
  Loan-schema = (branch-name, loan-number, amount)
  loan-number → amount branch-name
  Borrower-schema=(customer-name, loan-number)
- अब तीनों relation BCNF में है अत: हमें इसके बाद decompose करने की जरूरत नहीं पड़ेगी।

• BCNF हो सकता है कि dependency preservation की शर्तों को पूरा न करें।

- कुछ मामलों में, एक table जो BCNF नहीं है, उसे और decompose नहीं किया जा सकता, जिससे वह BCNF की शर्तों को पूरा कर सके और, dependencies को बिना बदले preserve रख सके जैसे की वो Original table में थी।
- उदाहरण के लिए, Functional dependencies के एक set (AB → C, C → B) को BCNF Schema में परिभाषित नहीं किया जा सकता।
- पहले वाले तीन Forms [1NF, 2NF, 3NF] के विपरीत BCNF कभी-कभी नहीं बन पाते हैं।
- नीचे दी गई table जो BCNF में नहीं है, को देखिए, जिसकी FD {AB → C, C → B} के Pattern में हैं।

Nearest Shop:

Person	*Shop Type*	*Nearest Shop*
Davidson	Optician	Eagle Eye
Davidson	Hairdresser	Snippets
Wright	Bookshop	Merlin Books
Fuller	Bakery	Doughy's
Fuller	Hairdresser	Sweeney Todd's
Fuller	Optician	Eagle Eye

प्रत्येक Person/shoptype मेल के लिए table हमें बताती है कि इस तरह की कौन सी shop उस व्यक्ति के घर के सबसे नजदीक है। हम अपनी सहूलियत के लिए यह मान लेते हैं कि एक shop एक type की है।

अब इस table की candidate key होगी

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

अब क्योंकि table के सभी attributes candidate key से संबंधित है table 3NF में है। परन्तु table BCNF नहीं है क्योंकि shop type attribute functionally 'nearest shop' attribute पर निर्भर करता है जो कि एक super key नहीं है। अब इसे अगर दो relation में decompose किया जाए:

Shop Near Person:

Person	*Shop*
Davidson	Eagle Eye
Davidson	Snippets
Wright	Merlin Books
Fuller	Doughy's
Fuller	Sweeney Todd's
Fuller	Eagle Eye

Shop:

Shop	*Shop Type*
Eagle Eye	Optician
Snippets	Hairdresser
Merlin Books	Bookshop
Doughy's	Bakery
Sweeney Todd's	Hairdresser

यहाँ shop near person की candidate key है: (person, shop) और shop table की candidate key है (shop). इसका Design जबकि BCNF के हिसाब से ठीक है परन्तु तब भी दुर्भाग्यवश, यह मान्य नहीं है क्योंकि यह हमें एक व्यक्ति के लिए एक तरह की बहुत सारी shops को डालने से नहीं रोकता। दूसरे शब्दों में इसकी Candidate key यह सुनिश्चित नहीं करती कि Functional dependency {person, shoptype} → {shop} को हमेशा माना जाएगा।

Multivalued Dependencies (बहुस्तरीय निर्भरताएँ)

- माना R कोई relation schema है or X और Y उसके disjoint अलग-अलग subset हैं इस तरह कि: $X \subseteq R, Y \subseteq R, X \cap Y = \varnothing$), और $Z = R \rightarrow XY$ है। एक relation r(R) $X \rightarrow Y$ को संतुष्ट करता है। अगर किन्हीं दो tuples (rows) r_1 and r_2 के लिए:
  - ➢ $t_1(X) = t_2(X)$, है तो r में कोई ऐसा t (row) होना चाहिए जो इस तरह हो:
    $$t_3(X) = t_1(X), t_3(Y) = t_1(Y), t_3(Z) = t_2(Z).$$
  - ➢ और समरूपता के कारण r में t_4 भी होना चाहिए इस तरह कि:
    $$t_4(X) = t_1(X), t_4(Y) = t_2(Y), t_4(Z) = t_1(Z).$$

	X	*Y*	*Z*
t_1	x_1	y_1	z_1
t_2	x_1	y_2	z_2
t_3	x_1	y_1	z_2
t_4	x_1	y_2	z_1

➢ MVD X-Y यह कहती है कि X और Y के बीच का संबंध, X और R-Y के संबंध पर निर्भर नहीं करता।

➢ उदाहरण के लिए, Employee table देखिए:

Employee:

Employee-name	*Project-name*	*Dependant-name*
Smith	X	John
Smith	Y	Ann
Smith	X	Ann
Smith	Y	John

➢ MVDs Employee name → Project-Name और Employee name → Dependent name इस relation पर लागू होती है।

➢ Smith नाम का Employee project X और Y पर काम करता है और John और Ann उस पर निर्भर करते हैं।

➢ अगर हम इस Relation की सिर्फ पहली दो row को ही store करें तो यह attributes के बीच के संबंधों को गलत तरीके से दिखाएगा।

➢ अगर Relation में MVDs है तो tuples में आपको repeated values मिलती रहेंगी। इस Employee

relation (table) में Project_name की value X और Y हर बार Dependent_name के साथ repeat हो रही है। जो कि अनावश्यक या बिना जरूरत का Data है।

- मुश्किल यह है कि employee Schema BCNF में है क्योंकि उस पर कोई FD लागू नहीं होती।
- **Trivial MVD:** अगर MVD X-Y सभी relation से संतुष्ट हो जाए जिनकी Schema में X और Y शामिल है तो यह trivial MVD कहलाती है।
  - $X \rightarrow Y$ trivial होगी जब भी $Y \subseteq X$ or $X \cup Y = R$
- अगर कोई relation r दी हुई MVD को संतुष्ट करने में नाकाम हो जाता है, तो एक relation r_1 जो MVD को संतुष्ट (Satisfy) करता है वह r में कुछ और tuples जोड़ने से मिलेगा।
  - MVD को 'tuple generating dependency' भी कहा जाता है।
  - अगर इसकी तुलना FD के साथ की जाए: तो कुछ tuples delete करनी पड़ेगी ताकि यह दी हुई FD को संतुष्ट कर सके।
- MVD को दो तरीकों से इस्तेमाल किया जा सकता है:
  - Relation को test करने के लिए कि वे किन्हीं दी हुई FD और MVD लिए legal (मान्य) है या नहीं
  - किसी relation में set पर constraints को specify (उल्लेख) करना।
- माना D एक FD और MVDs का set है तो D^+; D का Closure सभी FD और MVD का set होगा जो D द्वारा Logically लगाई गई है।
- D^+ को नीचे दिए गए नियमों से निकाला जा सकता है:

1. Reflexivity: अगर $Y \subseteq X$ तो $X \rightarrow Y$
2. Augmentation: अगर $X \rightarrow Y$ तो $WX \rightarrow Y$
3. Transitivity: अगर $X \rightarrow Y$ और $Y \rightarrow Z$ तो $X \rightarrow Z$
4. Complementation: अगर $X \rightarrow Y$ तो $X \rightarrow R\text{-}XY$
5. MV augmentation: अगर $X \rightarrow Y$ और $W \subseteq R$, $V \subseteq W$, तो $WX \rightarrow VY$
6. MV transitivity: अगर $X \rightarrow Y$ और $Y \rightarrow Z$ तो $X \rightarrow Z\text{-}Y$
7. Replication: अगर $X \rightarrow Y$ तो $X \rightarrow Y$
8. Coalescence: अगर $X \rightarrow Y$ और $Z \subseteq Y$, $W \subseteq R$, $W \cap Y = \varnothing$ $W \rightarrow Z$, तो $X \rightarrow Z$

Note: पहले तीन नियम Armstrong के axioms है।

4th Normal Form (4 NF):

- कोई relation schema R, 4NF में है (D के संदर्भ में), अगर प्रत्येक non-trivial MVD $X \rightarrow Y$ D^+ में होगी, और X, R के लिए super key होगी।
- **4NF vs BCNF**
  - 4NF BCNF से इसलिए अलग है क्योंकि यह F(FDs) की जगह D (FD + MVD) का इस्तेमाल करता है।
  - प्रत्येक 4NF Schema BCNF भी होगी।
    - Replication rule से $X \rightarrow Y, X \rightarrow Y$ को ही Implies करेगा। (replication का मतलब है बिल्कुल वैसे ही एक प्रति बना देना/Copy कर लेना।)

➢ अगर R, BCNF में नहीं है, तो एक Non-trivial FD X → Y मौजूद होगी जहाँ X Super key नहीं है। यहाँ पर R 4NF में भी नहीं हो सकता।

- उदाहरण के लिए, Employee (Employee-name, Project-name, Dependant-name) 4NF में नही है जब तक:
  - ➢ Employee-name → Project-name परन्तु Employee name कोई key नहीं है।
  - ➢ इसे 4NF में लाने के लिए Emp-proj (E-n, P-n) और Emp-dep (E-n, D-n) में Decompose करना पड़ेगा।
- उदाहरण के लिए Borrow (Loan#, C-name, Street, C-city) BCNF में है परन्तु 4NF में नहीं है, क्योंकि C-name → Loan# एक Non-trivial MVD है, जहाँ C-name Schema में कोई key नहीं है।
- इसे R_1 =(C-name, Loan#) और R_2=(C-name, Street, C-city) में decompose करने पर यह 4NF में आ जाएगा।
- 4NF के फायदे:
  - ➢ Tuples की संख्या को कम कर देता है।
  - ➢ Insert/Delete/Update के विसंगितयों को दूर कर देता है।
- FD और MVD की तुलना करने पर
  - ➢ अगर हमारे पास (a1,b1,c1,d1)∈r और (a1,b2,c2,d2)∈r है तो:
    - ♦ A → B का मतलब b1=b2
    - ♦ A → B का मतलब(a1,b1,c2,d2)∈r और (a1,b2,c1,d1)∈r

Join करने की निर्भरता और Fifth Normal Form

- अभी तक जो Normal Forms discuss हुए हैं, उनमें अगर Relation R दिए गए Normal Form के अनुरूप नहीं था तो हमने उसे दो relation में तोड़कर (decompose करके) उसे Normal Form के अनुरूप बनाया। परन्तु कुछ अलग मामलों में कभी कभी, किसी relation में duplicate information या update करने पर विसंगतियाँ आदि को समस्या होती है परन्तु वह समस्या को दूर करने के लिए दो relation में नहीं तोड़ा जा सकता। इस तरह के मामलों में हो सकता है, Relation को 3 या ज्यादा relation में decompose (तोड़ना) करना पड़े, 5 NF का इस्तेमाल करके।
- 5 NF Join-dependencies से निपटता है जो MVD का generalisation है। 5 NF का उद्देश्य है कि ऐसा relation मिले जिसे और आगे decompose ना किया जा सके। एक relation जो 5 NF में है, उसे कुछ छोटे relation से मिलाकर नहीं बनाया जा सकता।
- कोई Relation R Join dependency *(R_1, R_2, ..., R_n) को संतुष्ट करता है अगर R, (R_1, R_2, ..., R_n) के Join के बराबर हो (Join करने पर कोई loss ना हो) जहाँ प्रत्येक R_i, R के attributes का subset है।
- कोई relation 5 NF (या Project-Join normal form, PJNF) में है अगर Form की सभी Join dependencies *(R_1, R_2, ..., R_n) के लिए $R = R_1 \cup R_2 \cup R_3 .... \cup R_n$ होगा।
- प्रत्येक R_i, R की super key होगा। 5 NF का एक उदाहरण नीचे दिया गया relation है।

Department	*Subject*	*Student*
Comp. Sc.	CP1000	Sanjay
Mathematics	MA1000	Sanjay
Comp. Sc.	CP2000	Arun Kumar
Comp. Sc.	CP3000	Reena Rani
Physics	PH1000	Vijay
Chemistry	CH2000	Ajay

- यह relation कहता है कि computer science तीन subjects (CP1000, CP2000 और CP3000) Offer करती है जो अलग-अलग students ने लिए हुए हैं। किसी भी student ने सारे subject नहीं लिए हैं, और कोई Subject ऐसा नहीं जिसमें सभी विद्यार्थियों ने नामांकन कराया हो। इसलिए तीनों field Information को बताने के लिए जरूरी है।
- इस Relation में MVDs नहीं है, क्योंकि attribute student और subset independent नहीं है; ये एक दूसरे से संबंधित है और मिलाकर जरूरी सूचना प्रदान करते हैं। इसलिए इस relation को दो relation में decompose नहीं किया जा सकता, जैसे कि (dept, subject) और (Dept_student) बिना किसी Information Loss के। हालांकि, इस relation को हम तीन relation में बाँट सकते हैं।
  - (dept, subject), and
  - (dept, student)
  - (subject, student)

  अब जो Decomposition हुआ है Lossless (बिना Information loss) के होगा।
- पहले पढ़ी हुई Loan_infor_Schema को देखिए। मान लिजिए यह दिया हुआ की नीचे दी गई Join dependency इस पर लागू होती है

  *((loan-number,branch-name), (loan-number,customer-name), (loan-number,amount))

  तो यह 5NF में नहीं है, क्योंकि ये relation Schema super key को नहीं दर्शाते तो हमें इस relation को तीन relation में decompose करना पड़ेगा जैसा की Join dependency में दिया गया है।
  - (loan-number,branch-name),
  - (loan-number,customer-name), and
  - (loan-number,amount).

अभ्यास प्रश्न

1. Integrity Constraints के बारे में बताइए।
2. Referential Integrity के बारे में बताइए।
3. Functional dependencies को परिभाषित करें।
4. Normal Form क्या होते हैं?

5. Normalisation क्या है इसके क्या फायदे हैं?
6. First Normal Form [1NF] के बारे में बताइए।
7. Second Normal Form [2NF] के बारे में बताइए।
8. Third Normal Form [3NF] के बारे में बताइए।
9. BCNF Normal Form के बारे में बताइए।
10. Fourth Normal Form [4NF] के बारे में बताइए।
11. Join dependency और Fifth Normal Form [5NF] के बारे में बताइए।
12. Decomposition को समझाइए।
13. Multivalued dependencies (MVD) के बारे में बताइए।

❑❑❑

5

MYSQL/SQL

5.1 DATA DEFINITION LANGUAGE

DDL या Data definition Language में SQL की commands होती है, जिन्हें database schema को define करने के लिए इस्तेमाल किया जाता है। यह केवल database schema के विवरण (description) के साथ deal करता है और database objects के structures को create या modify करने के लिए इस्तेमाल किया जाता है।

DDL Commands के उदाहरण

- **CREATE** – यह database और उसके objects (जैसे table, index, functions, views, store procedure और triggers) को create करने के लिए इस्तेमाल किया जाता है।
- **DROP** – Database से object को delete करने के लिए किया जाता है।
- **ALTER** – database के structure को alter (बदलते) करने के लिए Use किया जाता है।
- **TRUNCATE** – किसी table से सभी records को हटाने के काम आता है, साथ ही records के लिए allocate किए गए सभी space भी remove हो जाते है।
- **COMMENT** – data dictionary में command add करने के लिए इस्तेमाल किया जाता है।
- **RENAME** – database के किसी object का नाम बदलने के लिए Use किया जाता है।

5.2 DATA MANIPULATION LANGUAGE

SQL की वह command जो database में data को manipulate करने के काम आती है, वह Data manipulation Language या DML से संबंधित होती है, इसमे अधिकतर SQL statements शमिल होती है।

DML के उदाहरण:

- **INSERT** – table में Data को Insert करने के काम आती है।
- **UPDATE** – किसी table में मौजूद data को update करने के काम आती है।
- **DELETE** – Database की किसी table से records को delete करने के काम आती है।

5.3 DATA CONTROL LANGUAGE (DCL)

DCL में commands जैसे की GRANT और REVOKE शामिल होती है जो data के access को control करने में काम आती है, और Rights (अधिकार) और permission (अनुमति) को भी control करती है।

DCL Commands के उदाहरण:

- **GRANT** – database को access करने के लिए user को privilege (विशेष अधिकार) प्रदान करने के लिए।
- **REVOKE** – Grand Command द्वारा दिए गए विशेषाधिकार वापस लेने के लिए।

5.4 SQL

SQL का पूरा नाम structure Query Language है। यह इक standard database Language है, जो relational database जैसे My SQL, Oracles, SQL server, Posture इत्यादि में Data को create maintain और retrieve करने के काम आती है। यह Relational Database system के लिए एक standard Language है।

MySQL, MS-Access, Oracle, mybar, Informix, Postgres और SQL जैसे RDBMS इसे अपनी standard Language के रूप में इस्तेमाल करते है।

SQL की Applications

SQL एक बहुत ज्यादा इस्तेमाल की जाने वाली Query Language है। इसके क्या-क्या उपयोग है उनसे कुछ नीचे दिए गए है।

- User को relational database Management system (RDBMS) का Data access करने की अनुमति प्रदान करती है।
- User को Data का वर्णत (Data को Describe) करने की अनुमति प्रदान करती है।
- User database से data को define या manipulate (बदला) कर सकता है।
- इसके साथ और Languages को भी इस्तेमाल किया जा सकता है, SQL modules, Libraries और per-compilers के द्वारा
- User को database table को बनाने (create) या हटाना (Drop) की सुविधा प्रदान करती है।
- Database में stored procedure और functions को create करने या view (देखने) की अनुमति प्रदान करती है।
- User को tables, procedures और views पर Permission लगाने की अनुमति प्रदान करते है।

SQL के फायदे

High Speed

SQL की Queries को बहुत बड़ी संख्या में records को Database से Quickly (जल्दी से) और efficiently (कुशलतापूर्वक) निकालने में इस्तेमाल किया जाता है।

Well Defined Standards Exist

इसके अन्दर अच्छी तरह से परिभाषित (define) किए हुई मानक है, जो ANSI और ISO द्वारा प्रमाणित किए गए है। Non-SQL database में ऐसे कोई स्पष्ट standards नहीं होते।

Coding की जरूरत नहीं होती

SQL की मदद से Database system को Manage (प्रबंधन) करने में बहुत सारे code लिखने की जरूरत नहीं होती। पहले SQL database, relational database के प्रयायी होते थे। जबसे Object Oriented DBMS का उद्भव होने लगा, तबसे relational database मे (इस्तेमाल) objects को store करने की क्षमता को बढ़ाना पड़ा।

SQL की हानियाँ

Interfacing की दिक्कते:

किसी SQL database में Interfacing ज्यादा जटिल होती है, code में कुछ लाइने जोड़ने से तुलना करने पर।

अधिकतर विशेषताएँ मालिकाना तरीके से लागू की गई है

हाँलाकि SQL database ANSI और ISO के standard के हिसाब से है, फिर भी कुछ database standard SQL का मालिकाना विस्तार (propriety extraction) करना चाहते है।

5.5 NAMING CONVENTIONS और GUIDELINES (नाम रखने के दिशानिर्देश)

Naming convention उन नियमों का set है जो Identifiers के लिए इस्तेमाल होने वाले character sequence को चुनते है, जो variables, types, functions और बाकी entities को source code and documentation (दस्तावेजो) में denote करते है।

Naming Conversion को इस्तेमाल करने का मुख्य उद्देश्य यह कि कोई भी नाम देखकर database के object के type (प्रकार) और purpose का पता कर सके।

अगर किसी object का नाम well-defined है, तो वह सारी information, जैसे object का प्रकार, objcet का काम, जिस table पर वह काम करता है, अपने आप बता देता है।

फायदे

एक बढ़िया Naming Convention के निम्नलिखित फायदे है:

1. Objects का नाम रखने का एक विशिष्ट तरीका बताता है, ताकि वह एक जैसे लगे या पता चले कि वह एक group से संबंधित है।
2. Source code को पढ़ने और समझने में आसान बना देते है।
3. किसी भी object के action name को स्पष्ट कर देते है कि यह क्या काम करेगा।
4. संभावित अस्पष्टता के मामले में स्पटष्ता को बढ़ाता है।

5. जो नए व्यक्ति पहली बार object को पढ़ रहे है, उनके लिए बेहतर समझ प्रदान करता है।
6. एक तरह के object को एक जगह या sequence में store रखता है ताकि कम समय में database से किसी object को निर्धारित किया जा सके।

Programming language में हम बहुत तरह के naming convention इस्तेमाल करते है। जैसे कि Pascal Case, Camel Case, Under-Score आदि। यह प्रत्येक व्यक्ति और organizations पर निर्भर करता है।

Table

Table को Database में data को store करने के लिए इस्तेमाल किया जाता है। Table के लिए निम्न तरीके की Naming Convation हो सकती है।

1. प्रत्येक table के नाम में "tbl" prefix (नाम से पहले use होने वाला) होना चाहिए।
2. "tbl" prefix के बाद अगला शब्द table का नाम होना चाहिए।
3. Table के नाम का पहला (letter) बड़ा (capital) होना चाहिए।
4. Table के नाम का आखिरी अक्षर Character "S" (या "es") के साथ खत्म होना चाहिए, जिससे पता चले कि यह बहुवचन है।
5. अगर किसी table के नाम में एक से ज्यादा शब्द है तो प्रत्येक शब्द का पहला अक्षर बड़ा (capital) होना चाहिए।

उदाहरण:

1. tblEmployees
2. tblBooks
3. tblStudents
4. tblEmployeeDetails

Views

View एक Virtual table होती है जो, SQL Statement के resultset पर आधारित होती है। इसमें table की तरह ही row और column होते हैं परन्तु ये कोई Data store नहीं करती। किसी view के field database की एक या ज्यादा tables के fields से ही लिए गए होते हैं।

1. प्रत्येक View का नाम इस syntax (वाक्य रचना) के हिसाब से होना चाहिए "Vw_<ResultSet>"
2. प्रत्येक view के नाम से पहले prefix "Vw_<Result Set>" आना चाहिए।
3. Result Set का पहला अक्षर बड़ा होना चाहिए।
4. Result set का आखिरी Character 's' या 'es' से खत्म होना चाहिए।
5. अगर Result set में एक से ज्यादा शब्द है तो प्रत्येक शब्द का पहला अक्षर बड़ा होना चाहिए।

उदाहरण:

1. vw_BookDetails
2. vw_EmployeeDetails
3. vw_OrderDetails

Primary Key Constraints

एक primary key table के प्रत्येक record की विशिष्ट पहचान करती है। इसके Naming convention नीचे दिए गए हैं:

1. एक Primary key का नाम इस syntax के अनुरूप होना चाहिए "PK_<TableName>"
2. प्रत्येक Primary key के नाम से पहले Prefix "PK_" आना चाहिए।
3. Table के नाम का पहला अक्षर बड़ा (Capital) होना चाहिए।
4. Table name का आखिरी अक्षर Character 'S' या 'es' से खत्म होना चाहिए, जिससे पता लग सके की यह बहुवचन है।
5. अगर Table name में एक से ज्यादा शब्द हो तो प्रत्येक शब्द का पहला अक्षर बड़ा होना चाहिए।

उदाहरण:

1. PK_Employees
2. PK_OrderDetails
3. PK_BookDetails

Foreign Key Constraints:

किसी Database table की Foreign key किसी दूसरी table में primary key होती है। Foreign key का मकसद Refrential Integrity को सुनिश्चित करना है।

Naming Convention for Foreign Key

1. एक Foreign key का नाम नीचे दिए गए syntax (वाक्य रचना के हिसाब से होना चाहिए।)

   "FK_<TargetTable>_<SourceTable>".

   यहाँ Target table वह table है जिसमें foreign key काम करती है और source table में वह Primary key के रूप में काम करती है।
2. प्रत्येक Foreign key के नाम से पहले prefix "FK_" आना चाहिए।
3. दोनों table के नाम का पहला अक्षर बड़ा होना चाहिए।
4. दोनों table नाम 'S' या "es" से खत्म होने चाहिए जिससे पता लेगे कि यह बहुवचन है।

उदाहरण:

1. FK_Employees_Projects
2. FK_Students_ContactNumbers
3. FK_Orders_Details

Default Constraints

Default constraint में, अगर किसी Column के लिए कोई value नहीं दी गई है तो उसमें Default value होती है–pre-set value. Default Constraint का naming convention है:

1. Default constraint को इस syntax का इस्तेमाल करना चाहिए: "DF_<TableName>_<ColumnName>.
2. प्रत्येक Default Constraint के नाम से पहले Prefix "DF_" आना चाहिए।
3. Table Name और Column Name दोनों के पहले अक्षर बड़े (capital) होने चाहिए।
4. Table name का आखिरी अक्षर 's' या 'es' से खत्म होनी चाहिए, यह बताने के लिए कि यह बहुवचन है।

उदाहरण:

1. DF_Employees_EmpName
2. DF_Employees_EmpSalary
3. DF_OrderDetails_OrderNumber

Unique Constraint

एक Unique constraint किसी database table के प्रत्येक record की Unique (अलग से) पहचान करता है। यह किन्हीं दो records के किसी विशेष column में same values नहीं डालने देता। unique constraint के लिए naming convention है:

1. Unique constraint हमेशा इस syntax (वाक्य संरचना) में होना चाहिए।

   "UQ_<TableName>_<ColumnName(s)>.
2. प्रत्येक Unique constraint के नाम से पहले "UQ" prefix आना चाहिए।
3. Table name और column name दोनों का पहला अक्षर बड़ा (capital) होना चाहिए।
4. Table का नाम "s" या "es" से खत्म होना चाहिए जिससे उसके बहुवचन होने का पता चलता है।
5. अगर Unique key में एक से ज्यादा column हो तो column का नाम under score (_) से अलग-अलग होना चाहिए।

उदाहरण:

1. UQ_Employees_EmpId_EmployeeName
2. UQ_OrderDetails_OrderNumber

Check Constraint

एक check constraint किसी column के लिए शर्तें निर्धारित करता है, जो सभी rows के लिए मानना जरूरी है। check constraint के लिए Naming convention है:

1. Check constraint हमेशा नीचे दिए गए syntax के हिसाब से होना चाहिए।

   "CHK_<TableName>_<ColumnName>.

2. प्रत्येक check constraint के नाम से पहले "chk" prefix अवश्य होना चाहिए।
3. Table के नाम और column के नाम का पहला अक्षर बड़ा (capital) होना चाहिए।
4. Table का नाम 's' या 'es' से खत्म होना चाहिए उसे बहुवचन दिखाने के लिए।

उदाहरण:

1. CHK_Employees_EmpSalary
2. CHK_Employees_Age
3. CHK_OrderDetails_OrderPrice

User-defined Functions (User द्वारा परिभाषित Function)

User defined functions SQL की statements का वह set है जो पहले Input लेता है, फिर उसे execute (उस पर काम) करता है और फिर परिणाम (Results) देता है। User_defined function के लिए Naming convention है:

1. User define function का syntax "fn_<ActionName>".
2. प्रत्येक User defined function के नाम से पहले "fn" prefix होना चाहिए।
3. Action name का पहला अक्षर बड़ा (capital) होना चाहिए।
4. अगर किसी Action name में एक से ज्यादा शब्द हैं तो प्रत्येक शब्द का पहला अक्षर बड़ा होना चाहिए।

उदाहरण:

1. fn_CalulateProfit
2. fn_CalculateTotal

User-Defined Stored Procedure

एक stored procedure पहले से ही संकलित (pre compiled) SQL statements का set है, जो बहुत से programms द्वारा Use किया जा सकता है।

User-defined stored procedure का Naming convention है:

1. इसका syntax है:

   "usp_<TableName><ActionName> ".
2. प्रत्येक user defined stored procedure के नाम से पहले "usp" prefix होना चाहिए।
3. प्रत्येक table और action के नाम का पहला अक्षर बड़ा (capital) होना चाहिए।
4. Table का 's' या 'es' से खत्म होना चाहिए।

उदाहरण:

1. usp_Employees_Insert
2. usp_OrderDetails_Delete

Indexes

यह उसी तरह काम करता है जैसे किताब से पहले Index होता है। किसी Chapter को ढूंढने के लिए हम Index से देखकर direct उस page no. पर पहुँच सकते हैं। इसी प्रकार 'Indexes' विशेष look up table होती है जिन्हें database का search engine use करता है, Data को तेजी से ढूंढ़ने के लिए।

Index का Naming convention है:

1. प्रत्येक Index का नाम इस syntax में होना चाहिए:

   IX_<TableName>_<Column(s)>.
2. प्रत्येक Index नाम के आगे prefix "IX_" होना चाहिए।
3. प्रत्येक table और column का नाम capital letter से शुरू होना चाहिए।
4. Table का नाम अक्षर 'S' या 'es' से समाप्त होना चाहिए, ताकि पता चले कि वह बहुवचन है।
5. अगर Index के नाम में एक से ज्यादा शब्द है तो प्रत्येक शब्द बड़े (capital) अक्षर से शुरू होना चाहिए और underscore(_) से अलग-अलग होना चाहिए।

उदाहरण:

1. IX_Employees_EmpId
2. IX_OrderDetails_OrderNo_OrderDate

Triggers

Triggers वह संग्रहित (stored) procedures होते हैं, जो अपने आप execute हो जाते हैं, जब कोई DML (Data Manipulation langues) command दी जाती है। Triggers के लिए Naming convention है:

1. प्रत्येक Trigger का नाम इस syntax में होने चाहिए–

   "TR_<TableName>_<ActionName>.
2. प्रत्येक trigger का नाम prefix "TR"_" से शुरू होना चाहिए।
3. प्रत्येक trigger का नाम और action name बड़े अक्षर (capital letter) से आरंभ होना चाहिए।
4. Table का नाम 's' या 'es' से खत्म होना चाहिए ताकि पता चले कि वह बहुवचन है।

उदाहरण:

1. TR_Employees_ AfterInsert
2. TR_OrderDetails_AfterUpdate
3. TR_Employees_InstedOfDelete

Naming Convention बनाते वक्त इन चीजों को छोड़ देना (Ignore) चाहिए।

1. किसी object के नाम के लिए पहले से define किए गए SQL के keywords का इस्तेमाल ना करें।
2. Objects के नाम के बीच space का इस्तेमाल ना करें।
3. एक तरह के object के नाम का pattern एक जैसा होना चाहिए। (एक तरह के object के लिए अलग-अलग नाम के तरीके इस्तेमाल ना करे।)
4. नाम में Numeric या special symbols का इस्तेमाल न करे (जैसे tbl_Employee 4, tbl_Employee@Asian)

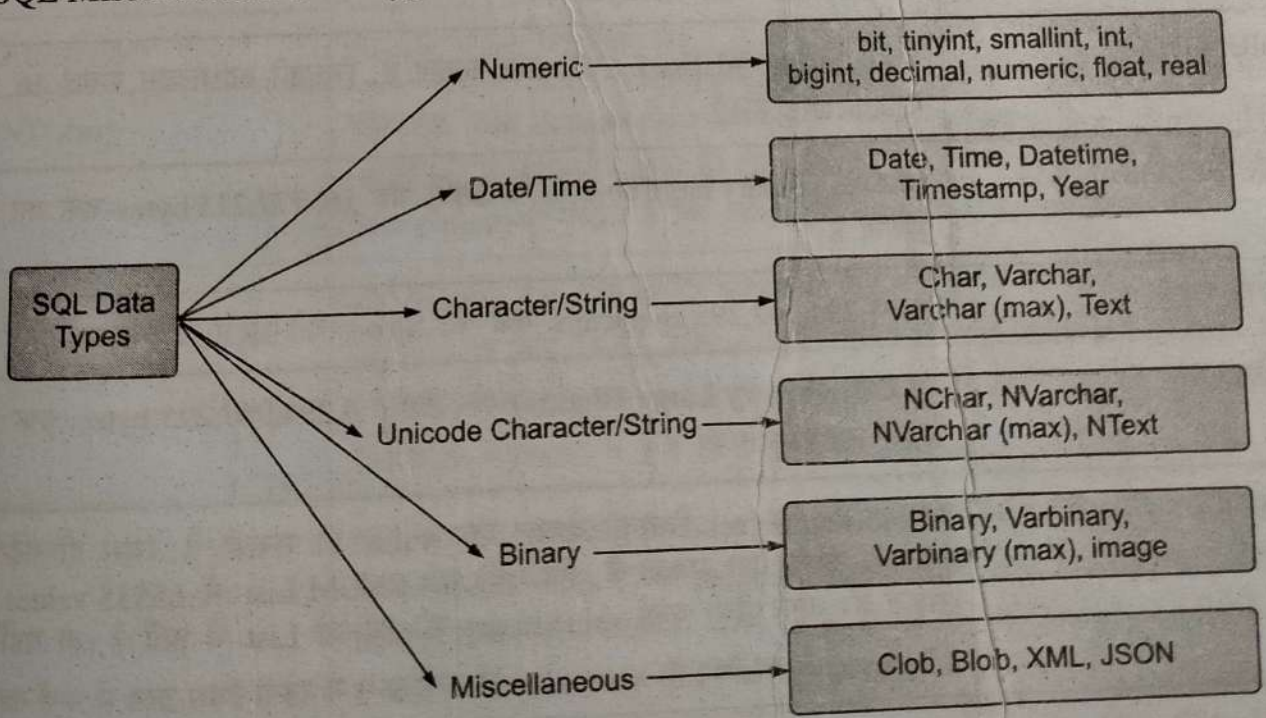
5.7 DATA TYPES

SQL के Data type वो attribute है जो (किसी भी object के data type के बारे में बताते हैं।)

प्रत्येक Column, variable और expression का SQL में एक संबंधित Data Type होता है। आप table बनाते (create) समय इन Data types का इस्तेमाल कर सकते हैं। आप table के किसी column के लिए एक Data type choose कर सकते हैं, अपनी जरूरत के हिसाब से।

SQL Server छः तरह के Data type को इस्तेमाल करने की सुविधा प्रदान करता है:

- String data types
- Numeric data types (सिर्फ संख्याओं के लिए)
- Date और Time के लिए data types
- SQL Unicode Character और String Data Types
- SQL Binary Data Types
- SQL Miscellaneous Data Types



String Data Types

Data type	*विवरण (Description)*
CHAR(size)	इसमें एक FIX length की string होती है। (जिससे Numbers, शब्द, special characters शमिल हो सकते हैं) string का size column की length को निर्धारित करता है–यह 0 से 255 तक हो सकता है। इसकी Default value 1 है।
VARCHAR(size)	यह VARIABLE length की string है (जिससे शब्द Numbers, संख्याएँ और special characters शामिल हैं।) इसकी range 0 से 65535 characters तक हो सकती है।
BINARY(size)	यह CHAR() के समान होता है, परन्तु सिर्फ binary byte strings को ही store करता है। इसका size column की length को bytes के रूप में निर्धारित करता है। Default value 1 है।
VARBINARY(size)	यह VARCHAR() के समान होता है, परन्तु केवल Binary byte strings को ही store करता है। इसका size column की maximum length को Bytes के रूप में निर्धारित करता है।
TINYBLOB	यह BLOBs (Binary Large Objects) के लिए इस्तेमाल होता है। अधिकतम length है 255 bytes
TINYTEXT	अधिकतम 255 characters की एक string को रख सकता है।
TEXT(size)	इसमे कोई String जिसकी अधिकतम Range 65, 535 bytes ही रख सकते हैं।
BLOB(size)	BLOBs (Binary large objects) के लिए जिनकी Max. Length 65, 535 bytes हो।
MEDIUMTEXT	किसी String को Hold (रख) कर सकता है, जिसकी अधिकतम लंबाई 16, 777, 215 Characters हो।
MEDIUMBLOB	BLOBs [Binary large objects के लिए] यह 16, 777,215 bytes तक का Data रख सकता है।
LONGTEXT	यह 4,294,967,295 characters तक की String को HOLD कर सकता है।
LONGBLOB	BLOBs (Binary Large Objects) के लिए। 4,294,967,295 bytes तक का Data hold कर सकता है।
ENUM(val1, val2, val3, ...)	ऐसा String object जिसकी केवल एक value हो सकती है, जिसे दी गई list में से choose किया जा सकता है, जिसे दी गई ENUM List में 65535 values तक डाल सकते हैं। अगर ऐसी कोई value insert की गई जो List में नहीं है, तो वहाँ blank value आ जाएगी। values list में उसी क्रम में रहेंगी जिस क्रम में उन्हें डाला गया है।
SET(val1, val2, val3, ...)	एक String object जिसकी values 0 या उससे ज्यादा हो सकती हे, जिससे दी गई Possible values की List में से चुन सकते हैं। एक List में आप 64 values तक डाल सकते हैं।

Numeric Data Types

Data type	*विवरण (Description)*
BIT(size)	यह bit-value प्रकार का data type है। एक value कितने bit की है यह इसके size को निर्धारित करती है। यह 1 से लेकर 64 तक value ले सकता है। Default value 1 है।
TINYINT(size)	यह बहुत छोटा Integer है। इसकी signed range –128 से +127 है। इसकी unsigned range 0 से 255 है। यह 1 byte का space लेता है (size parameter बताता है कि) अधिकतम Display width 255 है।
BOOL	यह दो Value लेता है 0 या non-zero 0 का मतलब False और Non-zero का मतलब true।
BOOLEAN	BOOL के समान ही होता है।
SMALLINT(size)	यह छोटा Integer type है। इसकी signed (–, + के साथ) range –32768 से + 32767 है और unsigned range 65535 है। इसे 2 byte के space की आवश्यकता होती है। (अधिकतम Display width 255 है)
MEDIUMINT(size)	यह medium Integer है। इसकी signed range –8388608 है और –8388607 से +8388607 है। और Unsigned range 0 से 16777215 है।
INT(size)	यह Medium Integer है। इसकी signed range –2147483648 से +2147483647 Unsigned range 0 से 4294967295 है। (अधिकतम Display width 255 है)
INTEGER(size)	यह Int (size) के समान है।
BIGINT(size)	यह एक बड़ा Integer type है। इसकी signed range –9223372036854775808 से +9223372036854775807 है। Unsigned range 0 से 18446744073709551615 है। Size parameter बताता है कि अधिकतम display width 255 है।
FLOAT(size, d)	यह floating point संख्या है। संख्या में कितने अंक है यह size specify करता है। और decimal के बाद कितने अंक (Digit) आ सकते हैं। यह 'd' parameter से specify (निर्दिष्ट) होता है। यह syntax My SQL 8.0.17 में शुरू हुआ था और आगे आने वाले MYSQL versions से हटा दिया गया।
FLOAT(p)	यह भी एक floating point number है। MySQL 'p' की value का यह जानने के लिए इस्तेमाल करता है कि resulting date type के लिए FLOAT या DOUBLE में से किसका प्रयोग होना चाहिए। अगर 'p' 0 से 24 है, तो Datatype FLOAT () बन जाता है और अगर 25 से 53 है तो यह DOUBLE () बन जाता है।
DOUBLE(size, d)	यह सामान्य size का floating point number है decimal के बाद कितने Digit आ सकते हैं। यह 'd' से निर्दिष्ट होता है। Size Parameter अंकों की सख्या बताता है।

Data type	*विवरण (Description)*
DOUBLE PRECISION (size, d)	
DECIMAL(size, d)	यह बिल्कुल exact fixed-point number है। इसमें कितने अंक है यह Size से specify (निर्दिष्ट होता है) या पता चलता है। Decimal के बाद कितने digit है यह 'd' से पता लगता है। Size के लिए अधिकतम Number 65 है। d के लिए अधिकतम Number 30 है। Size की Default value 10 है और d की default value 0 है।
DEC(size, d)	यह DECIMAL(size,d) के समान है।

Date and Time Data Types

Data type	*विवरण (Description)*
DATE	यह साल, महीने और दिन की Value को store करता है। इसका Format है: YYYY-MM-DD. इसकी range '1000-01-01' से '9999-12-31' है।
DATETIME(fsp)	यह Date और time का combination है। Format: YYYY-MM-DD hh:mm:ss है। जहाँ hh 'hour', mm, 'minute' और ss 'second' को बताता है। इसकी Range '1000-01-01 00:00:00' से लेकर '9999-12-31 23:59:59' तक है, Column की definition में DEFAULT और ON UPDATE command लगाने से, current date और time अपने आप update हो जाता है।
TIMESTAMP(fsp)	Time stamp में values seconds की संख्या में store होती है यह Unix epoch ('1970-01-01 00:00:00 UTC) में शुरू होता है इसकी range 1970-01-01 00:00:00 UTC है। DEFAULT CURRENT_TIMESTAMP और ONUPDATE CURRENT TIME STAMP के इस्तेमाल से Current date और time update कर सकते हैं।
TIME(fsp)	यह time को store करता है इसका FORMAT hh:mm:ss है। इसकी range '-838:59:59' से '838:59:59' तक है।
YEAR	Year एक 4 digit का format है। values 4-digit-format में होती है। 1901 से 2155 और 000 MySQL 8-0 2-digit के year format को support नहीं करती।

Unicode Character and String Data Types

Data type	*विवरण (Description)*
NCHAR	इसकी length fix होती है और अधिकतम 4000 character ले सकती है।
NVARCHAR	इसकी length variable होती है और अधिकतम 4000 character ले सकती है।
NVARCHAR(max)	यह variable length होती है और अधिकतम character store कर सकती है।
NTEXT	यह भी variable length storage है और अधिकतम 2GB का binary data store कर सकता है।

Binary Data Types

Data type	*Description*
BINARY	इसकी length fix होती है और अधिकतम length 8000 bytes होती है।
VARBINARY	यह Variable length storage है। अधिकतम Length 8000 bytes होती है।
VARBINARY(max)	Variable length storage जिसमें दी गई अधिकतम bytes store हो सकती है।
IMAGE	Variable length storage जिसमें अधिकतम 2GB का Binary data store किया जा सकता है।

और अलग तरह के (Miscellaneous) Data Type

Data type	*विवरण (Description)*
CLOB	Character large objects जो 2GB तक Data रख सकते हैं।
BLOB	Binary large objects के लिए Data type.
XML	XML data को store करने के लिए।
JSON	JSON Data को store करने के लिए।

5.8 CREATING TABLES (TABLES को बनाना)

किसी database में कोई नई table create करने के लिए CREATE TABLE statement का इस्तेमाल होता है।

Syntax:

```
CREATE TABLE table_name
{
   column1 datatype,
   column2 datatype,
   column3 datatype,
   ....
};
```

उदाहरण:

```
CREATE TABLE Student
{
   StudentID int,
   LastName varchar(255),
   FirstName varchar(255),
   Address varchar(255),
   City varchar(255)
};
```

StudenID column int type का होगा और केवल पूर्णांक (integer) ही store कर सकता है। Last name, First Name, Adress और City columns Varchar data type के है और characters को store कर सकते हैं। जहाँ किसी Field की अधिकतम length 255 character तक हो सकती है।

खाली "Students" table कुछ इस तरह दिखेगी:

StudentID	*LastName*	*FirstName*	*Address*	*City*

5.9 TABLES में VALUES डालना (INSERT करना)

INSERT INTO statement का इस्तेमाल किसी table में नया record डालने के लिए किया जाता है।

Syntax:

INSERT INTO STATEMENT को दो तरीकों से लिखा जा सकता है।

पहला तरीका

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

दूसरा तरीका

```
INSERT INTO table_name
VALUES (value1, value2, value3, ...);
```

उदाहरण:

INSERT INTO Student (StudentID, LastName , FirstName, Address, City,)

VALUES ('102', 'Singh', 'Ram', 'Ashok Nagar 21', 'Ghaziabad');

StudentID	*LastName*	*FirstName*	*Address*	*City*
102	Singh	Ram	Ashok Nagar 21'	Ghaziabad

5.10 TABLES को DELETE और UPDATE करना

UPDATE Statement का प्रयोग किसी tables में पहले से enter किए गए records को बदलने (संशोधित) के लिए किया जाता है।

Syntax:

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

उदाहरण:

```
UPDATE Student
SET LastName = ‹Sharma›, City= 'Delhi'
WHERE StudentID = 102;
```

StudentID	*LastName*	*FirstName*	*Address*	*City*
102	Sharma	Ram	Ashok Nagar 21'	Delhi

DROP TABLE statement का इस्तेमाल database की किसी मौजूदा table को drop (हटाने) करने के लिए जाता है।

Syntax:

```
DROP TABLE table_name;
```

उदाहरण:

```
DROP TABLE Student;
```

5.11 CONSTRAINTS को इस्तेमाल करना

SQL Constraints कुछ नियम होते हैं, जो column में data को insert, update और delete करते वक्त, इस्तेमाल किए जाते हैं ताकि column की अखंडता सुरक्षित रहे। Constraints उस समय ही उल्लिखित (specify) कर दिए जाते हैं, जब CREATE TABLE command से पहली बार table बनाई जाती है या फिर किसी मौजूदा table के structure को ALTER TABLE Statement से बदलते (modify करते) वक्त। SQL constraints को table के नियम लागू करने के लिए प्रयोग किया जाता है। अगर कभी नियमों का उल्लंघन होने से कोई action ठीक से पूरा नहीं होता तो constraint उस action को बंद कर देता है। (या पूरा नहीं होने देता)

कुछ constraint SQL की CREATE TABLE Statement के साथ इस्तेमाल किए जा सकते हैं।

SQL Constraints की सामान्य संरचना इस प्रकार है–

CONSTRAINT Keyword के बाद constraint का नाम आता है और उसके बाद column या columns की list आती है।

SQL CONSTRAINTS के प्रकार

SQL नीचे दिए गए constraints प्रदान करती है–

Constraint	*विवरण (Description)*
NOT NULL	इस constraint से यह सुनिश्चित होता है column Null value (खाली value) नहीं ले सकता।
UNIQUE	यह constraint यह सुनिश्चित करता है कि किसी columns के लिए प्रत्येक row की value अलग (unique) होनी चाहिए।

PRIMARY KEY	यह NOT NULL और UNIQUE constraint का combination है। यह सुनिश्चित करता है कि कोई विशिष्ट column या दो या उससे ज्यादा column का set किसी table के लिए ऐसी unique Identity (अलग पहचाना) हो, जिससे उस table में कोई विशेष record आसानी से और जल्दी मिल जाए। Primary key constraint यह भी ध्यान रखता है कि Primary key की value NULL नहीं होनी चाहिए।
CHECK	Check constraint यह सुनिश्चित करता है कि किसी column में store की गई value दी गई विशेष शर्तों को पूरा करती है या नहीं।
DEFAULT	अगर किसी column के लिए कोई value specify नहीं की गई है, तो default constraint उसके लिए default value प्रदान करता है।
FOREIGN KEY	Foreign Key constraint Data की Refrential Integrity को सुनिश्चित करता है। जब एक table के data को दूसरी table के data से match किया जाता है।

Syntax:

```
CREATE TABLE <table_name>
{
   column1    data_type[(size)]   constraint,
   column2    data_type[(size)]   constraint,
   ...
};
```

Parameters (मापदंड)

Name	*Description*
table_name	Table का नाम जिसमें data store होता है।
column1,column2	Table के columns के नाम
data_type	Data_type जैसे कि Char, Vchar, integers, decimal, date इत्यादि।
size	किसी table के column की अधिकतम length
constraint	Table या column के लिए लगाने वाले constraint

5.12 VIEWS

SQL में views एक तरह की virtual table table होती है। इसमें भी Database की अन्य real tables की तरह rows और columns होते हैं। हम Database की एक या अधिक tables से fields (columns) चुनकर किसी view को create कर सकते हैं। एक view किसी table की या तो सारी rows ले सकता है या फिर या कुछ विशेष rows भी ले सकता है, जो कि दी गई शर्तों पर निर्भर करता है।

हम CREATE VIEW statements का इस्तेमाल करके नया view बना सकते हैं। एक view किसी एक table या अनेक tables से बनाया जा सकता है।

Syntax (वाक्य-विन्यास)

```
CREATE VIEW view_name AS
SELECT column1, column2.....
FROM table_name
WHERE condition;
```

view_name: view का नाम जैसे ऊपर दिए गए उदाहरण में 'AS' view का नाम है।

table_name: वह table जहाँ से हमें columns लेने हैं।

condition: वह शर्त जिसके अनुसार उन columns के number or records या rows select की जाएँगी।

DELETING VIEWS (VIEWS को DELETE करना)

अभी हमने views को create करना सीखा है परन्तु अगर जो view हमने बनाया है, उसकी जब जरूरत नहीं है तो हम उसे Database से हटाना (delete करना) चाहेंगे। हम किसी भी view को Drop statement से delete या drop कर सकते हैं।

Syntax (वाक्य-विन्यास)

```
DROP VIEW view_name;
```

view_name: उस view का नाम जिसे आप delete करना चाहते हैं।

UPDATING VIEWS

किसी view को update करने के लिए कुछ शर्तों को पूरा करना जरूरी है। अगर इनमें से कोई एक भी शर्त पूरी नहीं होती; तो आप view को update नहीं कर पाएंगे।

1. View को बतमंजम करते वक्त जो SELECT Statement इस्तेमाल की गई थी, उसके साथ GROUPBY या ORDER BY clause नहीं होने चाहिए।
2. SELECT Statement में 'DISTINCT' Keywords नहीं होना चाहिए।
3. View में सभी NOT NULL Value होनी चाहिए।
4. View को nested queries या complex queries से नहीं बनाया गया हो।
5. View को कवेल एक ही table से बनाया गया हो। अगर view को अनेक tables से मिलाकर बनाया गया है, तो हम उसे update नहीं कर सकते।

हम किसी view से Fields (columns) को add (जोड़ने) या remove (हटाने) के लिए CREATE OR REPLACE VIEW Statement का प्रयोग करते हैं।

Syntax:

```
CREATE OR REPLACE VIEW view_name AS
SELECT column1,coulmn2,..
FROM table_name
WHERE condition;
```

Inserting a Row in a View (किसी View में row को insert करना)

(ध्यान दें हम यहाँ row की बात कर रहे हैं, Column या field को insert करने के लिए ऊपर वाली command CREATE OR REPLACE VIEW काम आएगी।)

हम किसी अपमू में row को उसी तरह Insert कर सकते हैं जैसे table में करते हैं। उसके लिए हम INSERT INTO STATEMENT का इस्तेमाल करते है।

Syntax:

```
INSERT INTO view_name(column1, column2 , column3,..)
VALUES(value1, value2, value3..);
```

view_name: view का नाम

अगर हमें Details view में सारा Data लेना है, तो SELECT*FROM Details view: का इस्तेमाल करते हैं।

```
SELECT * FROM DetailsView;
```

किसी View से row को Delete करना

किसी view से row को Delete करना उतना ही simple है जैसे table से row को delete करना। हम SQL की DELETE Statement का इस्तोमल करके किसी view से rows को delete कर सकते हैं।

इसके अलावा अगर वास्तविक table (जिससे view बना है) से rows delete कर देंगे तो बदलाव view में भी दिखाई देने लगेगा।

Syntax:

```
DELETE FROM view_name
WHERE condition;
```

जहाँ

view_name: जहाँ उस view का नाम है जिसकी rows delete करनी है।

condition: row को चुनने की शर्त है।

5.13 INDEXES

Index Schema का एक object होता है। यह server द्वारा, rows को एक pointer की मदद से आसानी से (retrieval) ढूंढने के लिए इस्तेमाल किया जाता है। यह Disk के I/O (Input/Output) को कम करता है, क्योंकि यह एक rapid path access method का प्रयोग करके data को आसानी से locate कर देता है।

एक Index SELECT QUERIES और where clause की speed को बढ़ा देता हे, परन्तु यह Data के Input को धीमा कर देता है, update और insert statements के साथ। Indexes को Data पर बिना प्रभाव डाले बनाया और हटाया जा सकता है।

उदाहरण के लिए अगर आप किसी किताब में से कोई विशेष topic पढ़ना चाहते है तो पहले हम Index को देखते हैं जिसमें सारे topics alphabetically या chapter wise दिए होते हैं कि कौन से page पर कौन सा topic है।

Index को बनाना (Create करना)

इसका Syntax है:

```
CREATE INDEX index
ON TABLE column;
```

जहाँ 'index', index को दिए जाने वाला नाम है और 'TABLE', table का नाम है जिसके लिए Index बनाया गया है और 'column' वह column है जिस पर index apply (लागू) होता है।

Multiple (एक से ज्यादा Columns के लिए)

```
CREATE INDEX index
ON TABLE (cloumn1, column2,.....);
```

Unique Indexes

```
CREATE UNIQUE INDEX index
ON TABLE column;
```

Unique indexes का इस्तेमाल table में मौजूद data की integrity को बनाए रखने के लिए किया जाता है, और साथ ही साथ बेहतर performance के लिए भी काम आता है। यह table में multiple values को नहीं डालने देता।

Index को कब बनाना चाहिए।

- जब एक column में बहुत ज्यादा Range की values हों।
- जब column में बहुत ज्यादा NULL Values ना हो।
- एक या ज्यादा Columns एक साथ बहुत बार Where या Join condition में इस्तेमाल हो रहे हों।

Index को कब नहीं बनाना चाहिए

- जब table बहुत छोटी हो।
- Columns किसी Query में condition की तरह बहुत ज्यादा इस्तेमाल ना होते हों।
- Column को बार-बार Update किया जाता हो।

किसी Index को हटाना

किसी Data dictionary से index को हटाने के लिए DROP INDEX command का प्रयोग होता है।

```
DROP INDEX index;
```

किसी Index को Drop करने के लिए (हटाने के लिए) या तो आप उस index के owner होने चाहिए या फिर आपके पास Drop ANY INDEX का विशेषाधिकार होना चाहिए।

Confirming Indexes

आप किसी विशेष table में User द्वारा या Server द्वारा दिए गए Indexes को Check कर सकते हैं और उनकी विशिष्टता का पता लगा सकते हैं।

```
select *
from USER_INDEXES;
```

5.14 SQL की COMMANDS

Describe

जैसा नाम बता रहा है, DESCRIBE किसी चीज का वर्णन करने के लिए इस्तेमाल किया जाता है। क्योंकि Database में हमारे पास Tables होती हैं हम DESCRIBE या DESC (दोनों एक ही है) Commands का इस्तेमाल किसी table के Structure को Describe (विवरण/वर्णन) करने के लिए करते हैं।

Syntax:

```
DESCRIBE one;
OR
DESC one;
```

Select

SELECT Command का इस्तेमाल Database से Data को Select करने के लिए किया जाता है। जो Data select होता है वह result table में store हो जाता है, जिसे result table में store हो जाता है, जिसे result-set कहा जाता है।

Syntax:

```
SELECT column1, column2, ...
FROM table_name;
```

यहाँ Column 1, Column 2... उन fields के नाम जिन्हें आप Data में से select (चुनना) चाहते हैं या निकालना चाहते हैं। अगर आप किसी table के सभी fields (columns) को select करना चाहते हैं तो नीचे दिए गए syntax का इस्तेमाल करें:

```
SELECT * FROM table_name;
```

Where Clause

Where Clause को records को filter (जो records चाहिए सिर्फ वही दिखेंगे) करने के लिए इस्तेमाल किया जाता है। यह सिर्फ उन्हीं records (rows) को निकालता है जो दी गई विशिष्ट शर्तों को पूरा करते हैं।

Syntax:

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Distinct Clause

SQL का District Clause. Select statement का इस्तेमाल करने के बाद चुने गए Result set में से Duplicates को हटा देता है।

Syntax:

```
SELECT DISTINCT expressions
FROM tables
[WHERE conditions];
```

Order By

Order By Keyboard का प्रयोग निकाले (select) किए गए Data को चढ़ते या उतरते क्रम में लगाने के लिए किया जाता है। अगर आप केवल Order By Keyword का प्रयोग करेंगे तो यह अपने आप Data को चढ़ते क्रम (ascending order) में लगा देगा। Records को उतरते क्रम में (Descending Order में) लगाने के लिए ORDER BY के साथ DESC keyword को भी लगाना पड़ेगा।

Syntax:

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

जहाँ–ASC, ascending (चढ़ते क्रम) और descending (उतरते क्रम) के लिए इस्तेमाल होता है। अगर हम ASC|DESC कुछ नहीं लगाएगे तो ORDER BY अपने आप Data को ascending order में लगा देगा।

Having

HAVING Clause को SQL में इसलिए Add किया गया है, क्योंकि WHERE Keyword को aggregate functions के साथ इस्तेमाल नहीं किया जा सकता।

Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

Logical Operations

Logical operators वह होते हैं जो या तो true होते हैं या false होते हैं। यह दी गई शर्तों या conditions को देखते कि वह true होगी या false होगी और फिर एक या इससे ज्यादा conditions को combine करके true या false value देते हैं। कुछ Logical operators हैं।

Operator	*Description (विवरण)*
AND	Logical AND दो Booleans की expression के रूप में तुलना करता है, और अगर दोनों True होते है तो true value देता है, अगर एक भी true नहीं है तो ये False value return करता है।
OR	Logical OR दो Booleans की expression के रूप में तुलना करता है और अगर दोनों में से कोई भी एक True है तो true value return करता है।
NOT	Not operation किसी एक Boolean को argument की तरह लेता है, और उसकी value को true है तो False में और False है तो true में बदल देता है।

कुछ विशेष operators

Operator	*Description (विवरण)*	*जिस पर Operate लागू होते हैं*
IN	IN Operator एक values के set में जिसमें बहुत सारी values को comma (,) से अलग-अलग किया होता है। इसमें से किसी value को check करता है और उस value से मिलती (matching) rows को Table से निकाल (Select कर) लेता है।	यह एक तरह की data types की values के set पर लागू होता है।
BETWEEN	यह किसी expression को एक Range में check करता है।	यह किसी भी Number पर, Character पर या Date time values पर भी लागू हो सकता है।

ANY	ANY किसी value को किसी List की values या किसी query से मिले Results से compare करता है और अगर उसे उसमें एक भी row मिल जाती है तो यह True Result देता है।	यह किसी List की value पर या किसी एक column set पर भी लागू हो सकता है।
ALL	यह किसी SELECT STATEMENT के सभी records को चुनने के के लिए इस्तेमाल होता है। यह किसी value को List या query के result की प्रत्येक value से compare करता है, All से पहले कोई comparision operator अवश्य होना चाहिए और अगर Query कोई row नहीं return करती तो इसका मूल्यांकन true होगा।	यह किसी List की value पर या किसी एक column set पर भी लागू हो सकता है।
SOME	SOME किसी value को list या query के result की प्रत्येक value से compare करता है और इसका मूल्यांकन true होगा अगर inner Querry के result में कम से कम एक row हो।	यह किसी List की value पर या किसी एक column set पर भी लागू हो सकता है।
EXISTS	EXISTS किसी subquery के result के अस्तिव को check करता है। EXISTS Subquery test करती है की कोई Subquery क्या कम से कम एक row निकालती है या नहीं। अगर कोई Data return नहीं होता तो operator 'False' result देगा।	यह किसी List की value पर या किसी एक column set पर भी लागू हो सकता है।

SQL Operators

SQL में Operator क्या होता है?

एक operator कोई शब्द या कोई character (कोई एक अंक या जैसे +, –, @, $ इत्यादि) हो सकता जिसे मुख्यत: SQL Statement के WHERE Clause में इस्तेमाल करने के लिए Reserve रखा जाता है, जिससे comparison और arithmetic (अंकगणित) के Operations (कार्य) किए जा सके। इन operators को SQL statement की conditions को specify करने के किया जाता है और यह एक statement की एक से ज्यादा conditions को जोड़ने का काम भी करते हैं।

- Arithmetic operators
- Comparison operators
- Logical operators
- Negate conditions के लिए इस्तेमाल होने वाले opeators

SQL के Arithmetic Operators

मान लीजिए कोई variable 'a' जिसकी value 10 है और variable 'b' जिसकी value 20 है तो–

Operator	*Description (विवरण)*	उदाहरण
+ (Addition) जोड़	Operator के दोनों तरफ की value को जोड़ देता है।	a + b होगा 30
– (Subtraction) घटाना	यह operator को right side की value को left side की value में से घटा देता है।	a – b होगा –10
* (Multiplication) गुणा करना	Operator के दोनों तरफ की values को गुणा कर देता है	a * b हमें देगा 200
/ (Division) भाग देना	Operators के Left side की value को right side की value से भाग दे देगा	b/a हमें देगा 2
% (Modulus)	यह left hand को value (इन्हें operand भी कहते हैं) को Right hand की value से भाग दे देता है और Remainder वापस कर देता है।	b %a का Result होगा '0'

SQL के Comparison Operators

मान लीजिए variable 'a' की value '10' है और 'b' की value '20' है तो–

Operator	*Description (विवरण)*	उदाहरण
=	यह check करता है कि operator के दोनों तरफ के operands की value बराबर है या नहीं, अगर हाँ तो condition true होगी।	(a = b) true नहीं है।
!=	यह check करता है कि दोनों operands बराबर हैं या नहीं, अगर नहीं तो condition true होगी।	(a != b) true है।
<>	यह check करता है कि दोनों operand बराबर हैं या नहीं, अगर नहीं तो condition true होगी।	(a <> b) true है
>	यह check करता है कि left node के operand की value right operand से ज्यादा है या नहीं, अगर हां तो condition true होगी।	(a > b) true नहीं है।
<	यह check करता है कि left operand Right operand की value से कम है या नहीं, अगर हां तो condition true होगी।	(a < b) true है।
>=	यह check करता है कि क्या left operand की value Right operand की value से बड़ी या बराबर है, अगर हाँ तो condition true होगी।	(a >= b) true नहीं है।
<=	यह check करता है कि क्या left operand की value right operand की value से कम या बराबर है। अगर हाँ तो condition ture होगी।	(a <= b) true है।

Operator	*Description (विवरण)*	उदाहरण
!<	यह check करता है कि क्या left operand की value Right operand से 'कम नहीं' (not less) है। अगर हाँ तो condition true होगी।	(a !< b) यहाँ true नहीं है। क्योंकि 10, 20 से 'कम नहीं' है।
!>	यह check करता है कि left operand की value, Right operand की value से 'ज्यादा नहीं' 'not greater है। अगर हाँ तो condition true होगी।	यहाँ (a !< b) true है क्योंकि 10, 20 से 'ज्यादा नहीं' है।

SQL के Logical Operators

यहाँ SQL के Logical operators की List दी गई है–

क्रमांक	*Operator और उनका Description (विवरण)*
1	**ALL:** ALL operator का इस्तेमाल किसी एक value को दूसरी values के set की सभी values से compare (तुलना) करने के लिए किया जाता है।
2	**AND:** AND operator किसी SQL statement के WHERE Clause में Multiple conditions (एक से ज्यादा शर्तों) को allow करता है।
3	**ANY:** ANY operator को किसी एक value को condition के हिसाब से किसी list की applicable value से compare करने के लिए इस्तेमाल किया जाता है।
4	**BETWEEN:** BETWEEN operator का इस्तेमाल दी गई Maximum (अधिकतम) और Minium (न्यूनतम) Values के बीच की values को दी गई values के set में से search करने के लिए किया जाता है।
5	**EXISTS:** EXISTS operator का इस्तेमाल यह seach करने के लिए किया जाता है कि दिए गए Criteria (मापदंड) के हिसाब से कोई row किसी table में है या नहीं (Exist करती है या नहीं)
6	**IN:** IN operator का इस्तेमाल किसी value को, Literal values की list से जो specified की गई है, से compare करने के लिए किया जाता है।
7	**LIKE:** LIKE operator का इस्तेमाल एक value को अपने जैसी value से compare करने लिए किया जाता है, wildcard operators का इस्तेमाल करके।
8	**NOT:** NOT operator किसी भी logical operator के मतलब को उल्टा (विपरीत) कर देता है। जिसके साथ इसे इस्तेमाल किया जाता है, जैसे: NOT EXISTS, NOT BETWEEN, NOT IN, इत्यादि। यह एक Negative operator है।
9	**OR:** OR operator का इस्तेमाल किसी SQL statement के WHERE Clause की Multiple conditions को combine (जोड़ने) करने के लिए किया जाता है।
10	**IS NULL:** NULL operator का इस्तेमाल किसी Value को NULL value के साथ compare करने के लिए किया जाता है।
11	**UNIQUE:** UNIQUE operator यह check करता है कि किसी table में कोई Duplicate row तो नहीं है। यह unique rows को search करता है।

JOIN

एक Join Clause को दो या उससे ज्यादा tables की rows को उनके बीच के Relation column के हिसाब से combine (जोड़ने) करने के लिए इस्तेमाल किया जाता है।

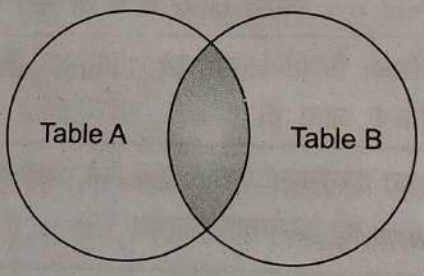
SQL में अलग-अलग प्रकार के JOIN होते हैं–

- **(INNER) JOIN:** यह वह records देता है जिनकी values दोनों tables में matching (एक जैसी) है।
- **LEFT (OUTER) JOIN:** यह left table के सारे Records और Right table के Matched Records को combine करता है।
- **RIGHT (OUTER) JOIN:** यह Right table के सभी records और left table के matched records को combine (जोड़ता) करता है।
- **FULL (OUTER) JOIN:** यह वो सारे Records देता है, जहाँ पर कोई match होता है, चाहे वो Left table में हो या Right table में।

1. **INNER JOIN:** INNER JOIN Keyword दोनों tables में Rows select करता है जब तक condition (शर्त) संतुष्ट होती रहती है। यह दोनों tables की rows को combine करके (जोड़कर) Result_set (table) बनाता है। यह उन Rows को जोड़ता है जो condition को संतुष्ट करती है मतलब जहाँ Common field एक जैसे हों।

Syntax:

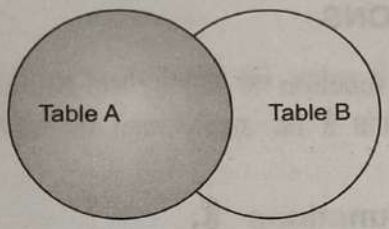
```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1
INNER JOIN table2
ON table1.matching_column = table2.matching_column;
```



2. **LEFT JOIN:** यह Join left side के table की सारी Rows को right side के table की matching rows (मिलती जुलती rows) के साथ जोड़ता है। वो Rows जिनसे Matching कोई rows Right side table में नहीं होती, उनके लिए result set NULL value होती है। LEFT JOIN को LEFT OUTER JOIN भी कहते हैं।

Syntax:

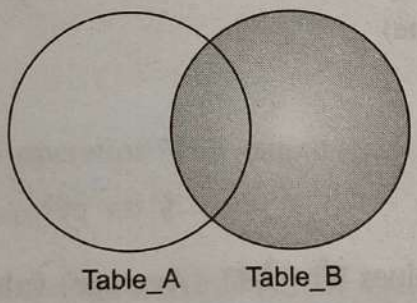
```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1
LEFT JOIN table2
ON table1.matching_column = table2.matching_column;
```



3. **RIGHT JOIN:** RIGHT JOIN, LEFT JOIN की तरह ही होता है। यह Join के Right Side के table की सभी Rows को left side के table की Matching rows को left side के table की Matching rows से जोड़ता है। वो rows जिनसे left side में कोई rows नहीं होती वो Matching left side में कोई rows नहीं होती वो result_set में NULL होती है। RIGHT JOIN को RIGHT OUTER JOIN भी कहते हैं।

Syntax:

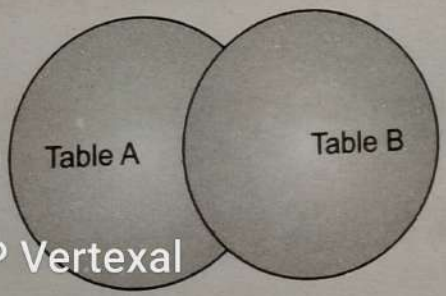
```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1
RIGHT JOIN table2
ON table1.matching_column = table2.matching_column;
```



4. **FULL JOIN:** Full LEFT Join और RIGHT JOIN दोनों के Result को combine करता है और Result_set बनाता है। Result_set में दोनों tables की row होती है। वो rows जिनके लिए कोई Matching नहीं होती, Result_Set में उनकी value NULL होती है।

Syntax:

```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1
FULL JOIN table2
ON table1.matching_column = table2.matching_column;
```



5.14 AGGREGATE FUNCTIONS

Database management में aggregate function वह होते हैं, जहाँ Multiple (बहुत सारी) rows की value को एक साथ इस तरह एक साथ group किया जाता है कि उसका ज्यादा महत्त्वपूर्ण अर्थ निकले।

अनेक प्रकार के Aggregate Functions हैं:

1. Count()
2. Sum()
3. Avg()
4. Min()
5. Max()

Count( )

Count function किसी field (column) की value की संख्या बताता है। यह numeric और non-numeric दोनों Data types पर काम करता है। सभी aggregate functions data पर काम करने से पहले 'NULL VALUES' को Exclude कर देते हैं, मतलब उन पर काम नहीं करते।

Syntax:

```
SELECT COUNT(column_name)
FROM table_name
WHERE condition;
```

AVG( )

AVG functions किसी column की values की औसत (Average) value बताता है। यह सिर्फ numerical data type (संख्याओं) पर काम करता है।

Syntax:

```
SELECT AVG(column_name)
FROM table_name
WHERE condition;
```

SUM( )

SUM function किसी column की सभी values का जोड़ (SUM) बताता है। यह सिर्फ संख्याओं (numbers) पर ही काम करता है। इससे जो Result (परिणाम) मिलता है उसमें NULL values नहीं होती।

Syntax:

```
SELECT SUM(column_name)
FROM table_name
WHERE condition;
```

MAX( )

जैसा कि इसका नाम बताता है, यह किसी table के field की value में से सबसे बड़ी value (maximum) को लौटाता है। यह MIN function का विपरीत होता है।

Syntax:

```
SELECT MAX(column_name)
FROM table_name
WHERE condition;
```

MIN( )

MIN function किसी table के field (column) की सभी values में से सबसे छोटी value (Minimum value) बताता है।

Syntax:

```
SELECT MIN(column_name)
FROM table_name
WHERE condition;
```

5.15 STRING FUNCTIONS

String functions Input String पर कुछ operation करके output string प्राप्त करने के लिए इस्तेमाल किए जाते हैं।

नीचे SQL के कुछ string functions दिए गए हैं–

1. **ASCII():** इस function को किसी character की ASCII Value को निकालने के लिए किया जाता है।

   उदाहरण: SELECT ascii('t'); इसका Result होगा

   Output: 116

   (ASCII: यहाँ ASCII का मतलब है American Standards code for Information Interchange)

2. **CHAR_LENGTH():** इस function को किसी शब्द (word) की लंबाई निकालने के लिए इस्तेमाल किया जाता है।

   उदाहरण: SELECT char_length('Hello!');

   Output: 6

3. **CHARACTER_LENGTH():** इस function को एक पूरी line की length निकालने के लिए इस्तेमाल किया जाता है।

   उदाहरण:SELECT CHARACTER_LENGTH('Asian Publishers');

   Output: 16

   (ध्यान रखें यहाँ Asian और Publisher के बीच के space को भी गिना गया है।)

4. **CONCAT():** इस function को दो शब्दों (words) या strings को जोड़ने के लिए इस्तेमाल किया जाता है।

   उदाहरण: SELECT 'Asian' || ' ' || 'Pubishers' FROM dual;

   Output: AsianPublishers

5. **CONCAT_WS():** यह function seperator साथ दो शब्दों (words) या strings को जोड़ने के लिए इस्तेमाल किया जाता है।

   उदाहरण: SELECT CONCAT_WS('_', 'Asian', 'Publishers');

   Output: Asian Publishers

6. **FIND_IN_SET():** इस function को कुछ symbols के set में से एक symbol को ढूँढने के लिए इस्तेमाल किया जाता है।

   उदाहरण: SELECT FIND_IN_SET('b', 'a, b, c, d, e, f');

   Output: 2

7. **FORMAT():** इस function को किसी Number को दिए गए Format के अनुसार Display करने के लिए इस्तेमाल किया जाता है।

   उदाहरण: Format("0.981", "Percent");

   Output: '98.10%'

8. **INSERT():** इस function को Database में Data Insert करने के लिए इस्तेमाल किया जाता है।

   उदाहरण: INSERT INTO database (book_name) VALUES ('database');

   Output: successfully updated

9. **LCASE():** इस function को किसी दी हुई string को lower case (छोटे अक्षरों) में बदलने के लिए इस्तेमाल किया जाता है।

   उदाहरण: LCASE ("AsianPublishers");

   Output: asianpublishers

10. **LEFT():** इस function का उपयोग दी गई string में से Left की तरफ से कुछ शब्द निकालने के लिए किया जाता है।

    उदाहरण: SELECT LEFT('asianpublishers', 5);

    Output: asian

    (इसने string में से left की तरफ के पाँच character निकाल दिए हैं)

11. **LENGTH():** इसे किसी शब्द की Length निकालने के लिए इस्तेमाल किया जाता है।

    उदाहरण: LENGTH(' AsianPublishers ');

    Output: 15

12. **LOWER():** इस function को UPPER CASE (बड़े अक्षरों) की string को Lower case (छोटे अक्षरों) में बदलने के लिए इस्तेमाल किया जाता है।

    उदाहरण: SELECT LOWER('ASIANPUBLISHERS');

    Output: asianpublishers

13. **REPEAT():** इस function को किसी string को बार-बार लिखने के लिए (जितनी बार दिया हुआ हो) इस्तेमाल किया जाता है।

    उदाहरण: SELECT REPEAT('ASIAN', 2);

    Output: ASIANASIAN

14. **REPLACE():** यह किसी string में से दी गई sub-string (उसका कोई हिस्सा) को हटा देता है।

    उदाहरण: REPLACE('123ASIAN123', '123');

    Output: ASIAN

15. **REVERSE():** यह किसी string के Alphabets (वर्णों) को उल्टा कर देता है।

    उदाहरण:SELECT REVERSE('asianpublishers');

    Output: 'srehsilbupnaisa'

16. **SPACE():** यह function दिए गए spaces (खाली स्थान) की संख्या को बताता है।

    उदाहरण: SELECT SPACE(7);

    Output: ' '

17. **STRCMP():** यह दो strings की तुलना (compare) करने के काम आता है।

    अगर string1 और string2 बराबर है तो यह Function result में '0' देगा।

    अगर string1, string2 से छोटी है (लंबाई में) तो यह function output में '-1' देगा

    अगर string1 string2 से बड़ी है तो यह '1' देगा।

    उदाहरण: SELECT STRCMP('google.com', 'asian publishers');

    Output: −1

18. **SUBSTR():** यह function किसी string में से दिए गए स्थान से शुरू करके, जितने chapter निकालने हैं, उनकी एक sub string बना देता है।

    उदाहरण 1: SUBSTR(' asianpublishers', 1, 5);

    Output: 'asian'

    यहाँ 1 स्थान से शुरू करके 5 character अलग कर दिए गए हैं।

    उदाहरण 2: SUBSTR('asianpublishers', 2, 5);

    Output: 'sianp'

19. **TRIM():** यह किसी string से दिए गए character/symbol को cut (हटाने) करने के लिए इस्तेमाल किया जाता है।

    उदाहरण: TRIM(LEADING '0' FROM '000123');

    Output: 123

20. **UCASE():** यह किसी string को UPPERCASE (बड़े अक्षरों) में बदलने के लिए इस्तेमाल किया जाता है।

    उदाहरण: UCASE ("asianpublishers");

    Output: ASIANPUBLISHERS

5.16 DATE TIME FUNCTIONS

SQL में नए लोगों के लिए Dates कुछ complicated (उलझा हुआ) Function होता है क्योंकि Date insert करने के लिए उसका Format table के Date format से Match करना चाहिए वरना हम Date Insert नहीं कर पाएंगें।

कई बार, Date की जगह Date time function (जिसमें Date के साथ time भी होता है) का इस्तेमाल किया जाता है।

My SQL में कुछ Default Date functions होते हैं:

1. **NOW():** यह current (अभी का) Date और time बताता है

   उदाहरण: SELECT NOW();

   Output: 2019-11-2 08:03:52

2. **CURDATE():** यह Current Date (आज की Date) बताता है।

   उदाहरण: SELECT CURDATE();

   Output: 2019-02-11

3. **CURTIME():** यह अभी (current time) बताता है।

   उदाहरण: SELECT CURTIME();

   Output: 08:05:15

4. **DATE():** यह किसी Date या Date/Time expression में से सिर्फ Date को निकालने के काम आता है। उदाहरण के लिए नीचे दी गई table 'Sample' के लिए:

Id	*Name*	*BirthTime*
120	Ram	1996-09-26 16:44:15.581

SELECT Name, DATE(BirthTime) AS BirthDate FROM Sample;

Output:

Name	*BirthDate*
Ram	1996-09-26

5. **EXTRACT():** यह किसी Date/Time में से केवल एक part को निकालता है। जैसे DAY, MONTH, WEEK इत्यादि।

Syntax: EXTRACT(unit FROM date);

हम unit में बहुत सारी units इस्तेमाल कर सकते हैं, पर जो ज्यादातर इस्तेमाल होती हैं वह हैं: MICROSECOND, SECOND, MINUTE, HOUR, DAY, WEEK MONTH, QUARTER, YEAR, इत्यादि।

और Syntax में 'date' एक कोई भी वैध (format के हिसाब से valid) expression होना चाहिए

उदाहरण: नीचे दी गई 'Sample' table देखिए

Id	*Name*	*BirthTime*
120	Ram	1996-09-26 16:44:15.581

SELECT Name, Extract(DAY FROM BirthTime) AS BirthDay FROM Sample;

Output:

Name	*BirthDate*
Pratik	26

5.17 NULL VALUES

SQL में NULL को Missing value को दिखाने के लिए इस्तेमाल किया जाता है। NULL value किसी table में वह value होती है जो किसी field (column) में खाली दिखाई देती है।

एक field जिसकी value NULL होती है, इसका मतलब उसकी कोई value नहीं होती।

यहाँ यह जानना बहुत जरूरी है कि, NULL value zero value (0) से अलग होती है और वह field जिसमें space होते हैं उससे भी अलग होती है।

Syntax:

table को बनाते वक्त (create) NULL value का जो मूल syntax है

उदाहरण:

```
CREATE TABLE CUSTOMERS
(
   ID INT  NOT NULL,
   NAME VARCHAR (20) NOT NULL,
   AGE  INT  NOT NULL,
   ADDRESS  CHAR (25) ,
   SALARY DECIMAL (18, 2),
   PRIMARY KEY (ID)
);
```

यहाँ NOT NULL यह बताता है कि उस column में हमेशा कोई ना कोई explicit (स्पष्ट) value (दिए गए Data type की) अवश्य होनी चाहिए। दो column (Address, SALARY), यहाँ हमने NOT NULL का इस्तेमाल नहीं किया है मतलब इन columns की value को खाली छोड़ा जा सकता है।

Isnull( )

SQL Server और MySQL में IS NULL FUNCTION के अलग अलग uses हैं। SQL Server में, IS NULL () Function को NULL values को बदलने के लिए इस्तेमाल किया जाता है।

Syntax:

```
SELECT column(s), ISNULL(column_name, value_to_replace)
FROM table_name;
```

Ifnull( )

यह function MySQL में तो है परन्तु Oracle और SQL Server में नहीं है। यह function दो arguments (तर्क) लेता है। पहला ये कि अगर पहला argument NULL नहीं है तो, function पहले argument को वापस कर देता है, वरना function दूसरे argument को वापस (return) कर देता है।

Syntax:

```
SELECT column(s), IFNULL(column_name, value_to_replace)
FROM table_name;
```

Coalesce( )

SQL में COLAESCE() Function इसके arguments में से पहला non-NULL expression निकाल देता है। अगर सभी arguments NULL होते हैं तो COLAESCE Function NULL value return करेगा।

Syntax:

```
SELECT column(s), CAOLESCE(expression_1,....,expression_n)
FROM table_name;
```

Nullif( )

NULLIF function के दो arguments होते हैं। अगर दोनों argument बराबर हों तो Function NULL return करता है और वरना पहला argument return करता है।

Syntax:

```
SELECT column(s), NULLIF(expression1, expression2)
FROM table_name;
```

अभ्यास प्रश्न

1. नीचे दी गई table से सभी salesmen की Information Display करने के लिए SQL Statement लिखिए:

Sample table: Salesman

Salesman_Id	*Name*	*City*	*Commission*
5001	James Hoog	New Youk	0.15
5002	Nail Knite	Paris	0.13
5005	Pit Alex	London	0.11
5006	Mc Lyon	Paris	0.14
5007	Paul Adam	Rome	0.13

2. इस string "This is SQL Exercise, Practice and solution" को दिखाने के लिए SQL Statement लिखिए।
3. तीन column में तीन numbers को Display करने के लिए Query लिखिए।
4. दो numbers 10 और 15 के जोड़ (SUM) को display करने के लिए query लिखिए।
5. किसी arithmetic expression के result को दिखाने के लिए Query लिखिए।
6. SELECT statement क्या है?
7. SQL में DISTINCT क्या है?
8. Student Table की ODD Rows को कैसे Display (दिखाया) जा सकता है।
9. अभी का (Current) date और time दिखाने के लिए Query लिखिए।
10. SQL में Date को कैसे Format किया जाता है?
11. Comparision operator को विस्तार से समझाइए।
12. SQL में किसी Date को YYYYMMDD format में कैसे दिखाएंगे?
13. Logical operators को उदाहरण के साथ समझाइए।
14. Table में Data को कैसे Update कर सकते हैं?
15. Aggregate function क्या है? विस्तार से समझाइए।
16. अलग-अलग प्रकार की Naming Conventions को विस्तार में समझाइए।
17. SQL के Data Types को समझाइए।
18. SQL में Indexes क्या होते हैं?
19. SQL की किसी Table को कैसे Delete (हटाया) किया जाता है?
20. DML, DDL और DCL को विस्तारपूर्वक समझाइए।
21. WHERE Clause को दो उदाहरणों के साथ समझाइए।

22. किसी table में values को कैसे Insert कर सकते हैं?
23. सभी arithmetic operators को उदाहरण के साथ समझाइए।
24. अलग-अलग प्रकार के JOINS को विस्तार पूर्वक समझाइए।
25. किन्हीं पाँच String functions को विस्तार से समझाइए।
26. NULL values क्या होती है?
27. Constraint क्या होते हैं? विस्तार में समझाइए।
28. SQL में VIEWS क्या होते हैं?
29. DISTINCT Clause को उदाहरण के साथ समझाइए।
30. नीचे दी गई table में से सिर्फ Name और commission के columns को display करने के लिए SQL Statement लिखिए।

Sample table: Salesman

Salesman_id	*Name*	*City*	*Commission*
5001	James Hoog	New York	0.15
5002	Nail Knite	Paris	0.13
5005	Pit Alex	London	0.11
5006	Mc Lyon	Paris	0.14
5007	Paul Adam	Rome	0.13

6

PL-SQL

6.1 परिचय (INTRODUCTION)

PL/SQL एक block strucured langauge है जो developers को SQL की ताकत के साथ-साथ Procedural Statements को भी इस्तेमाल करने की सुविधा प्रदान करती है। Block की सभी statements Oracle engine को एक साथ भेज दी जाती है, जिससे उसकी processing speed बढ़ जाती है और Data का traffic (बार-बार data का आना जाना, बार-बार Download/Upload) भी कम हो जाता है।

SQL से हानियाँ

- SQL programmers को conditions की checking, looping और branching की सुविधा नहीं देती।
- SQL की statement oracle engine में एक एक करके pass (भेजी जाती) होती है, इससे traffic बढ़ जाता है और speed कम हो जाती है।
- SQL के पास Data Manipulation करते वक्त errors को check करने की सुविधा नहीं है।

PL/SQL के विशेषताएँ (Features)

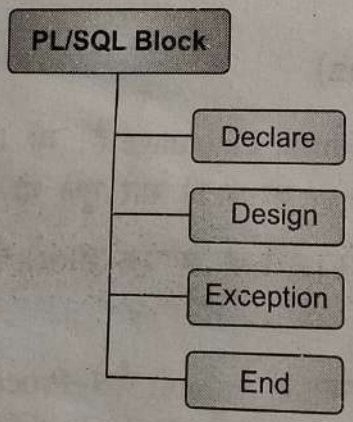
1. PL/SQL मूल रूप से एक procedural Language है, जो Decision making (निर्णय लेने की क्षमता), Iteration और Procedural langauge के बाकी सारे गुण भी प्रदान करती है।
2. PL/SQL Single command का इस्तेमाल करके एक Block की बहुत सारी Queries को एक साथ execute (process) कर सकती है।
3. हम PL/SQL की अलग-अलग इकाइयाँ (units) जैसे Procedures, Functions, Packages, triggers और type बना सकते हैं, जिन्हें किसी application में दोबारा इस्तेमाल करने के लिए store भी किया जा सकता है।
4. PL/SQL अपवादों (exceptions) को Handle करने के लिए एक विशेषता प्रदान करती है, जो PL/SQL के Block जिसे exception handling block कहा जाता है, में होती है।
5. PL/SQL में लिखी हुई applications को किसी ऐसे computer hardware या operating system तक ले जाया जा सकता है। जहाँ oracle चलता हो।
6. PL/SQL में बड़े पैमाने पर error checking (गलतियाँ ढूँढ़ना) भी की जा सकती है।

SQL और PL/SQL के बीच का अंतर

SQL	*PL/SQL*
SQL में एक Single query होती है जो DML और DDL operations को पूरा करती है।	PL/SQL Codes का एक Block होती है, जिसे पूरा Program block/procedure/function इत्यादि लिखने के लिए इस्तेमाल किया जाता है।
यह Declarative होती है, जो यह बताती है कि क्या करना है, यह नहीं कि कैसे करना है।	यह procedural langauge है, जो यह बताती है की चीजें कैसे करनी हैं।
यह एक single statement के रूप में कार्यान्वित (execute) होती है।	यह एक पूरे Block के रूप में कार्यान्वित (execute) होती है।
यह मुख्यत: Data को Manipulate (बदलने/इधर-उधर करने) करने के लिए इस्तेमाल होती है।	मुख्यत: किसी applicaiton को बनाने के लिए (create करने के लिए) इस्तेमाल होती है।
इसमें PL/SQL के codes का इस्तेमाल नहीं किया जा सकता।	क्योंकि यह SQL का ही एक विस्तार किया हुआ रूप है, यह SQL codes का इस्तेमाल कर सकती है।

PL/SQL Block का Structure

PL/SQL, SQL में Procedure Languages की तरह construct को जोड़कर उसका विस्तार करता है, जिसमें एक ऐसी Structure langauge बनती है जो SQL: से ज्यादा Powerful (उपयोगी/ताकतवर) होती है। PL/SQL की मूल इकाई Block होता है। PL/SQL के Program Blocks से बने हुए होते हैं, जो एक दूसरे के अंदर इस्तेमाल किए जा सकते हैं।



प्रत्येक Block Program में कुछ न कुछ Logical action जरूर करेगा। एक Block की संरचना (Structure) इस प्रकार होता है:

```
DECLARE
    declaration statements;
BEGIN
    executable statements
EXCEPTIONS
    exception handling statements
END;
```

- Declare section DECLARE Keyword के साथ शुरू होता है, जिसमें variables, constants, records को declare (घोषित करना) किया जाता है, जो Data को temporary रूप में store करते हैं। मूल रूप से इसमें PL/SQL के Identifiers की परिभाषा होती है। यह Code का वैकल्पिक (Optional) हिस्सा है।
- Execute section BEGIN से शुरू होता है और END से खत्म हो जाता है। यह जरूरी हिस्सा होता है और यहाँ किसी काम को करने के लिए Program का Logic लिखा जाता है जैसे– loops या conditional statements यह सभी DML commands, DDL commands को support करता है और SQL*PLUS के built-in function को भी support करता है।
- Exception Section EXCEPTION Keyword के साथ शुरू होता है। यह एक वैकल्पिक (Optional) हिस्सा है जिसमें वह statements होती है जो Run-time error (program execute होते वक्त त्रुटि आना) आने पर execute होती है। कोई भी exception (अपवाद) इस section द्वारा handle किया जाता है।

PL/SQL के identifiers

PL/SQL में कई तरह के Identifiers होते हैं जैसे variables, constants, procedures, cursors, striggers इत्यादि।

Variables

बहुत सारी अन्य Programming Langauges की तरह PL/SQL में भी variable को इस्तेमाल से पहले Declare करना जरूरी है। उनका एक valid नाम होना चाहिए और data type होना चाहिए।

Syntax

```
variable_name datatype [NOT NULL := value ];
```

उदाहरण:

```
SQL>  SET SERVEROUTPUT ON;
SQL>  DECLARE
   var1 INTEGER;
   var2 REAL;
   var3 varchar2(20);
BEGIN
   null;
END;
/
```

Output (Program का परिणाम)

```
PL/SQL procedure successfully completed.
```

व्याख्या (Explanation)

- **SET SERVEROUTPUT ON:** इसे dbms_output के Buffer (पहले से store किए हुए) को दिखाने के लिए इस्तेमाल किया जाता है।

- **var1 INTEGER :** यह variable का declaration है जिसका नाम Var1 है और इसका Data type integer है। और भी कई तरह के Data types को इस्तेमाल किया जा सकता है। जैसे float, Int, real smallint, long इत्यादि। यह SQL में इस्तेमाल होने वाले variables जैसे NUMBER (prec, scale), varchar, varchar 2 इत्यादि को भी support करती है।
- **PL/SQL procedure successfully completed:** यह code के सफलतापूर्वक पूरा होने पर display होता है।
- **Slash (/) का निशान END के बाद:** यह SQL*Plus को, block को execute करने का निर्देश देता है।

Variables को Initialise (आरंभ) करना

यहाँ भी variables को और Programming languages की तरह Initialise किया जाता है। उसका एक उदाहरण यहाँ देखते हैं–

```
SQL>  SET SERVEROUTPUT ON;
SQL>  DECLARE
        var1 INTEGER := 2 ;
      var3 varchar2(20) := 'Asian Publishers' ;
BEGIN
   null;
END;
/
```

Output:

PL/SQL procedure successfully completed.

Displaying Output (Output को दिखाना)

Output को DBMS_OUTPUT जो कि एक built-in Package है, से दिखाया (Display) जा सकता है, जो user को output दिखाने, Debugging INformaiton, के लिए और PL/SQL Blocks, Subprogram, Packages और triggers से message (संदेश) भेजने की सुविधा प्रदान करता है।

उदाहरण:

```
SQL>  SET SERVEROUTPUT ON;
SQL>  DECLARE
    var varchar2(40) := 'Asian Publishers' ;
BEGIN
   dbms_output.put_line(var);
END;
/
```

Output:

Asian Publishers

PL/SQL procedure successfully completed.

Comments का इस्तेमाल

बाकी Programming Languages की तरह PL/SQL में भी code के अंदर comments को डाला जा सकता है जिसका code पर कोई फर्क नहीं पड़ता यह सिर्फ कुछ Information देते हैं। PL/SQL में comment को डालने के लिए दो Syntax होते हैं।

- **Single Line Comment:** एक line के comment को डालने के लिए '.....' चिन्ह का इस्तेमाल किया जाता है।
- **Multi Line Comment:** एक से ज्यादा Line के comment को डालने के लिए /* और */ चिन्हों का इस्तेमाल किया जाता है।

Taking Input from User (User से Input लेना)

बाकी languages की तरह, PL/SQL में भी, हम User से Input लेकर उसे variable में store कर सकते हैं।

उदाहरण:

```
SQL>  SET SERVEROUTPUT ON;
SQL>  DECLARE
   -- taking input for variable a
   a number := &a;
   -- taking input for variable b
   b varchar2(30)  := &b;
BEGIN
   null;
END;
/
```

Output:

```
Enter value for a: 24
old  2: a number := &a;
new  2: a number := 24;
Enter value for b: 'Asian Publishers'
old  3: b varchar2(30) := &b;
new  3: b varchar2(30) := 'Asian Publishers';
PL/SQL procedure successfully completed.
```

PL/SQL Execution Environment (PL/SQL का काम करने का वातावरण)

PL/SQL engine एक Oracle engine में रहता है। Oracle engine single SQL statement के साथ-साथ बहुत सारी Statements के block को भी process कर सकता है। Oracle engine की आवश्यकता इसलिए पड़ती है कि एक बार PL/SQL block की कितनी भी SQL Statements को execute कर सकते हैं।

6.2 DATA TYPES

PL/SQL के variables, constants और Parameters का एक वैध (valid) data type होना चाहिए। जिससे storage का format, values की valid range constraints का पता चलता है। इस chapter में हम SCAER और LOB Data types पर ध्यान केंद्रित करेंगे। बाकी दो तरह के data type बाकी chapters में cover किए जाएँगे।

S. No	*Category & Description* (वर्ग और उनका विवरण)
1	**Scalar:** इसमें केवल एक value होती है और कोई Internal components नहीं होते जैसे– NUMBER, DATE या BOOLEAN
2	**Large Object (LOB):** यह large objects के pointers होते हैं जिन्हें बाकी data items से अलग store किया जाता है, जैसे– text, graphic images, video clips और sound wave forms
3	**Composite:** वो Data items जिनके अंदर भी components होते हैं जो अलग से access किए जा सकते हैं जैसे– Collections और Records
4	**Reference:** यह दूसरे Items के लिए Pointers होते हैं।

PL/SQL के Scalar Data Types और Subtypes

PL/SQL के Scaler data types और उनके Subtypes के विभिन्न प्रकार नीचे दिए गए हैं–

S. No	*Date Type* और उनके विवरण
1	**Numeric:** Numeric values (संख्यात्मक) वह होती है जिनपर अंकगणित के कार्य किए जा सकते हों।
2	**Character:** यह alphanumeric (अक्षर और अंक दोनों ले सकती है।) Value होती है, जो एक अकेले Character या String of characters को represent करती है।
3	**Boolean:** यह logical value होती है जिस पर logical operations किए जा सकते हैं।
4	**Datetime:** इस Data type में Date (दिनांक) और time (समय) को इस्तेमाल कियाजाता है।

PL/SQL में Data type के Subtypes (उपप्रकार) भी होते हैं। नीचे दी गई List में PL/SQL के Numeric data type और उनके sub-type दिए गए हैं।

S. No	*Data Type* और उनके विवरण
1	**PLS_INTEGER:** इस Integer की Range–2,147,483,648 से लेकर 2,147,483,647 तक होती है इसका size 32 bit होता है।
2	**BINARY_INTEGER:** इसकी Range– 2,147,483,648 से लेकर 2,147,483,647 तक होती है और इसका storage size 32 bit होता है।
3	**BINARY_FLOAT:** यह एक Single_precision IEEE-754 Format का floating-point number होता है।

4	**BINARY_DOUBLE:** यह एक Double_precision IEEE 754 Format का floating-number होता है।
5	**NUMBER(prec, scale):** यह Fixed-point या floating-point number होता है जिसकी absolute value की Range IE-130 (जो शामिल नहीं है) से लेकर 1.0E126 तक होती है एक Number variable '0' को भी represent कर सकता है।
6	**DEC(prec, scale):** यह ANSI का एक specific Fixed Point type है जो अधिकतम 38 decimal Digits तक ले सकता है। (preision 38 decimal तक होता है)
7	**DECIMAL(prec, scale):** यह IBM की specific fixed-point type है जिसका अधिकतम precision 38 decimal digits होता है।
8	**NUMERIC(pre, scale):** यह Floating type है जिसका अधिकतम Precision 38 decimal digit है।
9	**DOUBLE PRECISION:** ANSI का specific floating type है, जिसका maximum precision 126 binary digits है (लगभग 38 decimal digits)
10	**FLOAT:** ANSI और IBM का specific floating-point type है जिसका Maximum precision 126 binary digits है। (लगभग 38 decimal digits)
11	**INT:** ANSI का specific Integer type है, जिसका Maximum precision 38 decimal digits है।
12	**INTEGER:** ANSI का specific Integer type है, जिसका Maximum precision 38 decimal digit है।
13	**SMALLINT:** ANSI और IBM का specific Integer type है, जिसका Maximum precision 38 decimal digits है।
14	**REAL:** यह एक floating point type है जिसका Maximum precision 63 binary digits है (लगभग 18 decimal digits)

PL/SQL के Character Data Types और Subtypes

नीचे PL/SQL के predefined character types और sub-types को detail में दिया गया है।

S. No	*Date Type* और उनके विवरण
1	**CHAR:** यह Fixed-length की character string है, जिसका अधिकतम Size 32767 bytes है।
2	**VARCHAR2:** यह Variable length की character string है, जिसका अधिकतम size 32767 bytes है।
3	**RAW:** यह variable length की binary या byte string है, जिसका अधिकतम size 32767 bytes है, PL/SQL इसकी व्याख्या नहीं करता।
4	**NCHAR:** यह Fixed length की National Character string है, जिसका अधिकतम (Maximum) Size 32767 bytes है।

S. No	Date Type और उनके विवरण
5	**NVARCHAR2:** यह variable length की National character string है, जिसकी maximum size 32767 bytes है।
6	**LONG:** यह Variable length की character string है, जिसका Maximum size 32760 bytes है।
7	**LONG RAW:** यह variable length की binary या byte string है, जिसकी अधिकतम (maximum) size 32760 bytes है, PL/SQL इसकी व्याख्या नहीं करता।
8	**ROWID:** यह एक physical row identifier है, जो एक table में row का address है।
9	**UROWID:** यह एक universal row identifier है, (physical, logical या foreign row Identifier है।)

PL/SQL Boolean Data Types

BOOLEAN data type logical value को store करते है जो logical operations में काम आते हैं। logical value Boolean values होती है जो TRUE और FALSE और NULL हो सकती है।

SQL में BOOLEAN के बराबर कोई Data type नहीं होता। इसलिए Boolean values को निम्नलिखित में इस्तेमाल नहीं कर सकते।

- SQL statements में,
- Built-in SQL Functions में (जैसे TO_CHAR)
- SQL Statements से Invoked (लागू किए गए) PL/SQL Functions में।

PL/SQL के Datetime और Interval Types

DATE Data types को, Fixed_length के Date times (दिनांक और समय) को store करने के लिए इस्तेमाल किया जाता है, जिसमें दिन का समय seconds में होता है आधी रात से शुरू करके। इसकी Range है: January 1, 4712 BC से December 31, 9999 AD

Oracle Initialization parameters द्वारा set किया गया default format है NLS_DATE_FORMAT जैसे default format हो सकता है। 'DD-MON-YY', जिसमें दिन के लिए दो digit है, महीने के नाम के लिए तीन character हैं और आखिरी दो digit साल के लिए हैं। उदाहरण के लिए 01-Oct-12. प्रत्येक DATE में Century, Year, Month, day, hour, minute और Second शामिल होते हैं। नीचे दी गई table प्रत्येक Field की Valid value को दर्शाती है:

Each DATE includes the century, year, month, day, hour, minute, and second. The following table shows the valid values for each field:

Field Name	Valid Datetime Values	Valid Interval Values
YEAR	-4712 से लेकर 9999 ('0' को छोड़कर)	कोई भी Non zero integer
MONTH	01 से 12	0 से 11

DAY	01 से 31 (Month और साल के हिसाब से, Calender के Rule के हिसाब से)	कोई भी Non-zero Integer
HOUR	00 से 23	0 से 23
MINUTE	00 से 59	0 से 59
SECOND	00 से 59, 9(n), जहाँ 9(n) time के fractional seconos का precision है।	0 से 59, 9(n) जहाँ 9(n) Interval fractional seconds का Precision है।
TIMEZONE_HOUR	-12 से 14 (इस Range में Daylight savings time changes भी है)	–NA–
TIMEZONE_MINUTE	00 से 59	–NA–
TIMEZONE_REGION	यह Dynamic Performance view में मिलता है V$TIMEZONE_NAMES	–NA–
TIMEZONE_ABBR	यह Dynamic Performance view में मिलता है V$TIMEZONE_NAMES	–NA–

PL/SQL के Large Object (LOB) Data Type

Large Object (LOB) data types large objects के लिए होते हैं जैसे कि text, graphics, images, video clips and sound waveforms. LOB data types इस तरह के बड़े object के लिए प्रभावशाली, जल्दी से और piecewise (एक-एक piece के हिसाब से) data access करता है। नीचे कुछ PL/SQL के LOB Datatypes दिए गए है।

Data Type	*Description (विवरण)*	*Size*
BFILE	यह Large binary objects को Database से बाहर operating system की files में store करने के काम आता है।	इसका size system पर निर्भर करता है। परन्तु 4GB से ज्यादा नहीं हो सकता।
BLOB	Database के अंदर Large binary objects करने के लिए इस्तेमाल किया जाता है।	8 से 128 Terabytes (TB)
CLOB	Character data के बड़े Blocks को Database में store करने के लिए इस्तेमाल किया जाता है।	8 से 128 TB
NCLOB	NCHAR Data के बड़े blocks को Database में store करने के लिए इस्तेमाल किया जाता है।	8 से 128 TB

PL/SQL के User-Defined Subtypes

एक Subtype किसी data type का subset होता है। Subtype वह सभी operation कर सकता है, जो उसका base data type (जिसमें से वह निकला है) कर सकता है, और उसकी कुछ Valid values का subset होता है।

PL/SQL में NULLs

PL/SQL ने package standard में बहुत से subtypes को पहले से परिभाषित (per-defined) किया हुआ है। PL/SQL में NULL Value Missing (छूटा हुआ) या unknown data (जिसकी value पता नहीं हो) को दर्शाती है, और ये ना तो Integer Integer होते हैं, ना ही character या कोई और विशेष Datatype यहाँ ध्यान दें की NULL किसी empty data string या Character 'O' के बराबर नहीं होता। किसी चीज के लिए NULL को assign किया जा सकता है परन्तु वह किसी भी चीज के बराबर नहीं हो सकता, स्वयं के भी नहीं।

6.3 OPERATORS

Operator वह चिन्ह (SYMBOL) होता है जो compiler को कि कौन सा Mathematical (गणितीय) या Logical operation करना है। PL/SQL में बहुत सारे built-in operators होते हैं जैसे कुछ नीचे दिए गए हैं:

- Arithmetic operators (अंकगणित वाले)
- Relational operators
- Comparison operators (तुलना करने वाले)
- Logical operators
- String operators

Arithmetic Operators (अंकगणित के Operators)

नीचे दी गई table में कुछ arithmetic operators दिए गए हैं। यहाँ हम एक variable a को '10' मान लेते हैं और variable b को '5' मान लेते हैं अब:

Operator	विवरण	उदाहरण
+	यह दोनों operands (operator के दोनों तरफ की value) को जोड़ देता है।	a + b हमें 15 देगा
-	यह दूसरे operand को पहले में से घटा देता है।	a – b हमें 5 देगा
*	यह दोनों operands को गुणा कर देता है।	a * b हमें 50 देगा
/	यह numerator (अंश) को denominator (हर) से भाग देता है।	A / B हमें 2 देगा
0**	यह exponential operator है यह एक पर दूसरे की power चढ़ा देगा।	A ** B का मतलब a^b और यह हमें 100000 देगा।

Relational Operators (आपस में संबंध बताने वाले)

यह दो expression या values की तुलना करता है और Boolean result (True या Not True) देता है। नीचे दी गई table में कुछ logical operators दिए गए हैं। यहाँ मान लीजिए A की value '10' है और B की value '20' है तो-

Operator	विवरण	उदाहरण
=	यह check करता है कि दोनों operands की value बराबर है या नहीं, अगर हाँ तो condition true होगी।	(A = B) यहाँ NOT TRUE है।
!= <> ~=	यह check करता है कि दोनों operands की value बराबर है या नहीं, अगर बराबर नहीं है तो condition true होती है।	यहाँ (A != B) TRUE है।
>	अगर बाएँ तरफ के operands की value दाएँ तरफ के operand से ज्यादा है, तो condition true होगी।	यहाँ (A > B) (10 > 20) NOT TRUE है।
<	अगर बाएँ तरफ के operand की value दाएँ तरफ के opeand से कम है, तो condition True होगी।	यहाँ (A < B) TRUE है।
>=	अगर बाएँ तरफ के operand की value दाएँ तरफ के operand से ज्यादा या बराबर है तो condition True होगी।	यहाँ (A >= B) NOT TRUE है
<=	अगर बाएँ तरफ के operand की value दाएँ तरफ के operand से कम या बराबर होगी तो condition TRUE होगी।	यहाँ (A <= B) TRUE है।

Comparison Operators

Comparison operators, एक expression की दूसरे तुलना करने के लिए इस्तेमाल किए जाते हैं। इसका Result TRUE, FALSE या NULL ही होगा।

Operator	विवरण	उदाहरण
LIKE	Like operator किसी character, string या CLOB की किसी pattern से तुलना करता है और TRUE देता है अगर pattern Match कर जाता है। वरना FALSE देता है।	अगर 'Zara Ali' LIKE 'Z% A_i' यहाँ True देगा तो 'Nuha Ali' like 'Z% A_i' False देगा।
BETWEEN	BETWEEN बताता है कि कोई value किसी विशेष range में आती है या नहीं। x BETWEEN A AND B का मतलब है कि x >= a और x <= b.	अगर x = 10 तो x BETWEEN 5 and 20 हमें true देगा और x between 11 and 20 हमें False देगा।
IN	IN operator यह check करता है कि कोई value किसी SET की member है या नहीं। X IN (Set) का मतलब है कि X उस SET का कोई member है।	अगर x = 'm' है तो X IN ('a', 'b', 'c') देगा false परन्तु X IN ('m', 'n', 'o') हमें देगा True.
IS NULL	यह True देना अगर operand NULL है, और false देगा अगर operand NULL नहीं है।	अगर x = 'm', है तो 'x is null' हमें False देगा।

Logical Operators

नीचे दी गई table में PL/SQL के Logical operators दिए गए हैं। ये सभी Boolean Operands पर काम करते हैं और Boolean Result ही देते हैं। मान लीजिए किसी variable की value 'true' है और variable B की value 'false' है।

Operator	*विवरण*	*उदाहरण*
and	अगर दोनों operands true है तो ही condition true होगी।	यहाँ (A and B) false है।
or	यह logical operator OR को बुलाता है। अगर दोनों operands में से कोई एक भी true है तो condition true होगी।	यहाँ (A or B) true है।
not	यह operands के logical state को उलट देता है। अगर कोई condition true है तो NOT operator उसे false कर देता है और अगर कोई condition False है तो NOT operator उसे true कर देता है।	not (A and B) यहाँ true हो जाएगा।

PL/SQL Operator Precedence (कौन सा Operator पहले operate करेगा कौन सा बाद में)

यह किसी expression में terms की grouping बताता है। यह बताता है कि किसी expression को कैसे evaluate किया जाता है।

किसी operator का precedence order दूसरे operator से पहले हो सकता है, जैसे Multiplication operator (*) additional operator (+) से पहले operate करेगा तो उसक precedence order पहले है।

उदाहरण के लिए: X = 7 + 3 * 2, यहाँ X की value 13 होगी, क्योंकि, * operator पहले operate करेगा।

नीचे table दी गई है जिसमें जिस operator की precedence value ज्यादा है वह सबसे ऊपर है, और जिसकी precedence value कम है वह सबसे नीचे है। किसी expression में, higher precedence value वाला operator पहले कार्य करता है।

Operators का Precedence ऐसे होता है– =, <, >, <=, >=, <>, !=, ~=, ^=, IS NULL, LIKE, BETWEEN, IN.

Operator	*Operation*
**	exponentiation
+, -	identity, negation
*, /	multiplication, division
+, -, ||	addition, subtraction, concatenation
comparison	
NOT	logical negation
AND	conjunction
OR	inclusion

6.4 USER DEFINED FUNCTION

एक user द्वारा define किया गया function, **CREATE FUNCTION** statement से बनाया जाता है। **CREATE OR REPLACE PROCEDURE** statement के लिए सरल syntax नीचे दिया गया है।

```
CREATE [OR REPLACE] FUNCTION function_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
RETURN return_datatype
{IS | AS}
BEGIN
   < function_body >
END [function_name];
```

जहाँ

- *function-name* function का नाम उल्लिखित (specify) करता है।
- [OR REPLACE, option किसी पहले से मौजूद function को बदलता है।
- Parameter की List में Parameter के नाम, mode और type के बारे में details होती है। IN उस value को दिखाता है, जो बाहर से आती है और OUT उस parameter को दिखाता है, जो procedure से बाहर की value को वापस करता है।
- Function में return statement होना आवश्यक है।
- RETURN Clause उस datatype के बारे में बताता है, जिसे आप function से Return करने वाले हैं।
- Function-body के अंदर execute होने वाला हिस्सा होता है।
- अगर हमें stand alone function create करना है तो IS keyword की जगह AS Keyword का इस्तेमाल होगा।

Calling a Function (किसी function को बुलाना/इस्तेमाल करना)

किसी Function को create करते समय, आपको यह परिभाषित करना पड़ेगा कि function क्या करता है। किसी function को इस्तेमाल करने के लिए आपको उस function को कोई निर्धारित जो पूरा करने के लिए बुलाना (call करना) पड़ेगा। जब कोई program किसी function को call करता है तो program का control उस बुलाए गए function को transfer हो जाता है।

जब function अपने काम को पूरा कर देता है और उसकी Return statement कार्यान्वित (execute) हो जाती है या जब उसकी **last end statement** आ जाती है, तो यह program का control main program को वापिस कर देता है।

किसी function को call करने के लिए, आपको required parameters को function के नाम के साथ बताना पड़ेगा और अगर function कोई value return करता है तो आप उस value को store कर सकते हैं।

PL/SQL के Recursive Functions (पुनरावृत्ति/दोहराने वाले)

हमने यह देखा है कि कोई program या subprogram किसी दूसरे sub-program को call कर सकता है। जब कोई subprogram खुद को ही call कर लेता है तो उसे recursive call कहते हैं और process को **recursion** कहते हैं।

इस concept को स्पष्ट करने के लिए, किसी number का factorial निकालते हैं। किसी number का factorial होता है।

```
n! = n*(n-1)!
   = n*(n-1)*(n-2)!
     ...
   = n*(n-1)*(n-2)*(n-3)... 1
```

नीचे दिए हुए program किसी दिए गए number का factorial निकालता है, खुद को बार-बार program में call करके:

```
DECLARE
   num number;
   factorial number;

FUNCTION fact(x number)
RETURN number
IS
   f number;
BEGIN
   IF x=0 THEN
      f := 1;
   ELSE
      f := x * fact(x-1);
   END IF;
RETURN f;
END;

BEGIN
   num:= 6;
   factorial := fact(num);
  dbms_output.put_line(' Factorial '|| num || ' is ' || factorial);
END;
/
```

Output:

Factorial 6 is 720

PL/SQL procedure successfully completed.

6.5 PL/SQL की FLOW STATEMENT का CONTROL

PL/SQL में तीन तरह की control statements होती हैं–

- **Conditional selection statements:** जो अलग-अलग Data values के लिए अलग-अलग satements run करती है। IF और CASE condition selection statements होती है।
- Loop statements: यह अलग-अलग Data values की series के लिए एक ही statement run करती है। Loop statements में LOOP, FOR LOOP, और WHILE LOOP होते हैं। EXIT Statement LOOP के आखिर में control को दे देती (transfer कर) देती है। CONTINUE Statement LOOP के एक Iteration (round) से से निकलकर उसके दूसरे Iteration (round) को control देती है। दोनों EXIT और CONTINUE में, एक optional WHEN Clause होता है जहाँ आप condition लगा सकते हैं।
- **Sequential Control statements:** यह PL/SQL Programming के लिए बहुत ज्यादा महत्त्वपूर्ण नहीं होती। GOTO एक sequential control statement होती है जो किसी विशेष condition तक ले जाती है और NULL वह statement है, जो कुछ नहीं करती।

Conditional Selection Statements

Conditional Selection Statements, IF और CASE, अलग-अलग data values के लिए अलग-अलग statements को run करते हैं। IF Statements एक या ज्यादा Statements के sequence को या तो run करती (चलाती) है, या skip कर (छोड़) देती है, जो दी गई condition (शर्त) पर निर्भर करता है।

IF Statement निम्न प्रकार की होती है–

- IF THEN
- IF THEN ELSE
- IF THEN ELSIF

CASE Statement Conditions (शर्तों) के sequence (श्रंखला) में से चुनकर, उसके हिसाब से Statement run करती है।

CASE Statement के प्रकार होते हैं–

- Simple, जो एक अकेले empression को evaluate (मूल्यांकन करना) करती है और कुछ संभावित values से इसकी तुलना करती है।
- Searched, जो एक से ज्यादा conditions का मूल्यांकन करती है और जो सबसे पहली true value होती है उसे चुनती है।

एक CASE Statement जब उपयुक्त होती है जब प्रत्येक alternative के लिए अनेक तरह के actions लेने होते हैं।

IF THEN Statement

IF THEN Statement का structure होता है।

```
IF condition THEN
   statements
END IF;
```

अगर **condition** (शर्त) true होगी तो **statements** run करेगी वरना IF Statement कुछ नहीं करेगी।

उदाहरण:

```
DECLARE
PROCEDURE P
{
   sales  NUMBER,
   quota  NUMBER,
   emp_id NUMBER
}
IS
   bonus    NUMBER := 0;
   updated  VARCHAR2(3) := 'No';
BEGIN
   IF sales > (quota + 200) THEN
   bonus := (sales - quota)/4;
   UPDATE employees
   SET salary = salary + bonus
   WHERE employee_id = emp_id;
   updated := 'Yes';
END IF;
DBMS_OUTPUT.PUT_LINE
{
   'Table updated?  ' || updated || ', ' ||
   'bonus = ' || bonus || '.'
};
END p;
BEGIN
   p(10100, 10000, 120);
   p(10500, 10000, 121);
END;
/
Result:
No, bonus = 0.
Yes, bonus = 125.
```

IF THEN ELSE Statement

IF THEN ELSE Statement का structure नीचे दिया गया है–

```
IF condition THEN
   statements
```

```
ELSE
  else_statements
END IF;
```

अगर condition की value true है तो statement run होगी वरना else_statement run होगी।

उदाहरण:

```
DECLARE
PROCEDURE p
(
   sales  NUMBER,
   quota  NUMBER,
   emp_id NUMBER
)
IS
   bonus  NUMBER := 0;
BEGIN
   IF sales > (quota + 200) THEN
      bonus := (sales - quota)/4;
   ELSE
      bonus := 50;
END IF;
DBMS_OUTPUT.PUT_LINE('bonus = ' || bonus);

UPDATE employees
SET salary = salary + bonus
WHERE employee_id = emp_id;
END p;
BEGIN
   p(10100, 10000, 120);
   p(10500, 10000, 121);
END;
/
```

Result:

bonus = 50

bonus = 125

Nested IF THEN ELSE Statements

```
DECLARE
PROCEDURE p
(
   sales  NUMBER,
   quota  NUMBER,
    emp_id NUMBER
)
IS
   bonus  NUMBER := 0;
BEGIN
   IF sales > (quota + 200) THEN
      bonus := (sales - quota)/4;
   ELSE
   IF sales > quota THEN
      bonus := 50;
   ELSE
      bonus := 0;
   END IF;
END IF;

DBMS_OUTPUT.PUT_LINE('bonus = ' || bonus);
UPDATE employees
SET salary = salary + bonus
WHERE employee_id = emp_id;
END p;
BEGIN
   p(10100, 10000, 120);
   p(10500, 10000, 121);
   p(9500, 10000, 122);
END;
/
```

Result:

bonus = 50

bonus = 125

bonus = 0

IF THEN ELSIF Statement

IF THEN ELSIF Statement का structure नीचे दिया गया है–

```
IF condition_1 THEN
   statements_1
ELSIF condition_2 THEN
   statements_2
{
   ELSIF condition_3 THEN
   statements_3
}...
{
   ELSE
   else_statements
}
END IF;
```

The IF THEN ELSIF statement उस पहली statements को run करती है, जिसके लिए **condition** (शर्त) true होती है। बाकी बची हुई conditions की मूल्यांकन (evaluation) नहीं किया जाता।

अगर कोई भी condition true नहीं है, तो else_statement run होती है, अगर वो होती है, तो वरना IF THEN ELSIF Statement कुछ नहीं करती।

उदाहरण:

```
DECLARE
PROCEDURE p (sales NUMBER)
IS
   bonus  NUMBER := 0;
BEGIN
   IF sales > 50000 THEN
      bonus := 1500;
   ELSIF sales > 35000 THEN
      bonus := 500;
   ELSE
      bonus := 100;
END IF;
DBMS_OUTPUT.PUT_LINE
{
   'Sales = ' || sales || ', bonus = ' || bonus || '.'
};
END p;
BEGIN
```

```
    p(55000);
    p(40000);
    p(30000);
END;
/
```

Result: *(परिणाम)*

Sales = 55000, bonus = 1500.

Sales = 40000, bonus = 500.

Sales = 30000, bonus = 100.

एक single (अकेली) IFTHEN ELSIF Statement समझने में बहुत आसान होती है, एक logically equivalent nested IF THEN Else Statement की अपेक्षा:

- IF THEN ELSIF statement

```
IF condition_1 THEN statements_1;
   ELSIF condition_2 THEN statements_2;
   ELSIF condition_3 THEN statement_3;
END IF;
```

- Logically equivalent nested IF THEN ELSE statements

```
IF condition_1 THEN
   statements_1;
ELSE
  IF condition_2 THEN
    statements_2;
  ELSE
     IF condition_3 THEN
        statements_3;
     END IF;
  END IF;
END IF;
```

CASE Statement

एक सामान्य Case Statement का structure है–

```
CASE selector
WHEN selector_value_1 THEN statements_1
WHEN selector_value_2 THEN statements_2
...
WHEN selector_value_n THEN statements_n
{
ELSE
```

```
    else_statements
  }
  END CASE;]
```

Selector एक expression होता है। (आमतौर पर एक single variable होता है) प्रत्येक Selector_Value या तो literal (शाब्दिक) या expression (अभिव्यक्ति) होता है। (पूरे syntax के लिए, << CASE Statements>> देखे)

एक सामान्य CASE Statement उस पहली statement को run करती है जिसके लिए Selector_value Selector के बराबर होती है। बाकी conditions का मूल्यांकन नहीं किया जाता। अगर कोई Selector_value Selector के बराबर नहीं होती, CASE Statement else Statements को करती है (अगर else statement है तो) वरना एक पहले से defined expression 'CASE_NOT_FOUND' को run करती है।

उदाहरण:

```
DECLARE
  grade CHAR(1);
BEGIN
  grade := 'B';
  CASE grade
    WHEN 'A' THEN DBMS_OUTPUT.PUT_LINE('Excellent');
    WHEN 'B' THEN DBMS_OUTPUT.PUT_LINE('Very Good');
    WHEN 'C' THEN DBMS_OUTPUT.PUT_LINE('Good');
    WHEN 'D' THEN DBMS_OUTPUT.PUT_LINE('Fair');
    WHEN 'F' THEN DBMS_OUTPUT.PUT_LINE('Poor');
    ELSE DBMS_OUTPUT.PUT_LINE('No such grade');
  END CASE;
END;
/
```

Result (परिणाम):

Very Good

LOOP Statements

Loop statements अलग-अलग value की श्रंखला (series) के लिए एक जैसी Statements run करती है। LOOP Statements कई प्रकार के हैं जैसे–

- Basic LOOP
- FOR LOOP
- Cursor FOR LOOP
- WHILE LOOP

वो Statement जो LOOP से बाहर निकालती है वो है:

- EXIT
- EXIT WHEN

वो Statement जो loop के वर्तमान Cycle से बाहर निकालती है और नए cycle को शुरू करती है वह है:

- CONTINUE
- CONTINUE WHEN

EXIT, EXIT WHEN, CONTINUE, and CONTINUE WHEN statements loop में कहीं भी आ सकती है, परन्तु LOOP के बाहर इस्तेमाल नहीं हो सकती है। Oracle "GO TO statement" जो पूरे loop या current iteration (वर्तमान cycle) से बाहर निकलने के बाद control को loop के बाहर के statement को transfer कर देते हैं, की जगह इन statements को इस्तेमाल करने की सलाह देता है। मूल loop statement का structure नीचे दिया है:

```
[ label ] LOOP
   statements
END LOOP [ label ];
```

Loop के प्रत्येक Iteration (cycle) के साथ, statement run होती है और loop के top को control वापस कर देती है। एक अनंत (Infite) Loop को रोकने के लिए statement या exception statements को loop से exit होना जरूरी है।

EXIT Statement

EXIT statement Loop के वर्तमान iteration से बिना शर्त बाहर निकाल देती है (exit कर देती है) और control को या तो वर्तमान loop के अंत में या किसी और संलग्न (enclosed) loop को transffer कर देती है।

CONTINUE Statement

CONTINUE statement loop के current iteration (एक cycle) से बिना शर्त निकलकर control को loop के दूसरे Iteration को transfer कर देता है।

FOR LOOP Statement

FOR LOOP Statement एक या ज्यादा Statements को RUN करता है, जबकि LOOP का index एक specified range में होता है। इस statement का structure है।

```
[ label ] FOR index IN [ REVERSE ] lower_bound..upper_bound LOOP
   statements
END LOOP [ label ];
```

REVERSE के बिना, index की value Lower_bound से शुरू होती है, और LOOP के प्रत्येक Iteration के साथ एक (संख्या '1') से बढ़ती रहती है, जब तक Value upper_bound तक नहीं पहुँच जाती। अगर Lowqer_bound की value upper_bound से ज्यादा होगी तो, statement कभी भी run नहीं करेगी।

REVERSE के साथ, index की value upper_bound से शुरू होती और Loop के प्रत्येक iteration के साथ एक से कम होती रहती है जब तक वह lower_bound तक न पहुँच जाए। अगर upper_bound से कम है तो statement कभी भी run नहीं होगी।

एक EXIT, EXIT When, CONTINUE और CONTINUE WHEN Statements LOOP को या वर्तमान iteration को जल्दी भी खत्म कर सकती है।

उदाहरण:

```
BEGIN
  DBMS_OUTPUT.PUT_LINE ('lower_bound < upper_bound');

  FOR i IN 1..3 LOOP
    DBMS_OUTPUT.PUT_LINE (i);
  END LOOP;

  DBMS_OUTPUT.PUT_LINE ('lower_bound = upper_bound');

  FOR i IN 2..2 LOOP
    DBMS_OUTPUT.PUT_LINE (i);
  END LOOP;

  DBMS_OUTPUT.PUT_LINE ('lower_bound > upper_bound');

  FOR i IN 3..1 LOOP
    DBMS_OUTPUT.PUT_LINE (i);
  END LOOP;
END;
/
```

Result (परिणाम):

lower_bound < upper_bound

1

2

3

lower_bound = upper_bound

2

lower_bound > upper_bound

FOR LOOP Statement में EXIT WHEN या CONTINUE WHEN Statement

मान लीजिए आप FOR LOOP से exit कर जाते है, जब कोई निश्चित परिस्थिति उत्पन्न होती है। तब आप condition को EXIT WHEN Statement में रख सकते हैं, FOR LOOP के अन्दर

उदाहरण:

```
DECLARE
  v_employees employees%ROWTYPE;
```

```
    CURSOR c1 is SELECT * FROM employees;
  BEGIN
    OPEN c1;
    -- Fetch entire row into v_employees record:
    FOR i IN 1..10 LOOP
      FETCH c1 INTO v_employees;
      EXIT WHEN c1%NOTFOUND;
      -- Process data here
    END LOOP;
    CLOSE c1;
  END;
  /
```

WHILE LOOP Statement

WHILE LOOP Statement एक या ज्यादा Statements run करती है जब कोई condition true होती है। उसका structure ऐसा होता है।

```
[ label ] WHILE condition LOOP
  statements
END LOOP [ label ];
```

अगर condition true होगी तो, statement Run होगी और control वापिस loop के Top में चला जाएगा। जहाँ condition का दोबारा मूल्यांकन होगा कि condition true है या नहीं। अगर condition true नहीं होगी तो, control WHILE LOOP की Statements के बाद वाली Statement के पास चला जाएगा। LOOP को INFINTE (अनंत, खत्म नहीं होने वाला) LOOP में जाने से रोकने के लिए, LOOP के अंदर की Statement को condition को या तो FALSE या NULL करना पड़ेगा।

उदाहरण:

```
DECLARE
  done  BOOLEAN := FALSE;
BEGIN
  WHILE done LOOP
    DBMS_OUTPUT.PUT_LINE ('This line does not print.');
    done := TRUE;  -- This assignment is not made.
  END LOOP;

  WHILE NOT done LOOP
    DBMS_OUTPUT.PUT_LINE ('Hello, world!');
    done := TRUE;
  END LOOP;
END;
/
```

Result:

Hello, world!

Sequential Control Statements

IF और LOOP Statements से भिन्न, sequential control statements, GOTO और NULL PL/SQL Programming के लिए उतनी महत्त्वपूर्ण नहीं है।

GOTO statement जो आपको एक विशेष statement तक ले जाती है, उसकी जरूरत कभी-कभी पड़ती है।

NULL Statement, जो कुछ नहीं करती, बस readability को बेहतर बनाती है, condition statement के action और meaning (मतलब) को और clear बनाकर।

GOTO Statement

GOTO Statement बिना किसी शर्त के किसी Lable को Control transfer कर देती है। यह Lable अपने दायरे में सबसे अलग होना चाहिए और ये एक executable statement या PL/SQL Block से पहले होना चाहिए। जब RUN की जाती है, तो GOTO statement Label Statement या block को control transfer कर देती है।

उदाहरण:

```
DECLARE
  p  VARCHAR2(30);
  n  PLS_INTEGER := 37;
BEGIN
  FOR j in 2..ROUND(SQRT(n)) LOOP
    IF n MOD j = 0 THEN
      p := ' is not a prime number';
      GOTO print_now;
    END IF;
  END LOOP;

  p := ' is a prime number';

  <<print_now>>
  DBMS_OUTPUT.PUT_LINE(TO_CHAR(n) || p);
END;
/
```

Result (परिणाम):

37 एक prime number है।

NULL Statement

NULL Statement सिर्फ अगली Statement को control pass करती है। कुछ Languages इस तरह के [illegible] भी कहते हैं।

NULL Statement के लिए target प्रदान करना।

- GOTO Statement के लिए target प्रदान करना।
- Condition statement के मतलब और action को clear बनाकर, readability को बेहतर बनाना
- Place holders और Sub programs बनाना।
- यह दिखना कि आपको पता है Possibility क्या है, परन्तु कोई action लेना अनिवार्य नहीं है।

6.6 PROCEDURES/STORED PROCEDURES

PL/SQL में stored procedure या सामान्य रूप से, procedure एक PL/SQL का block है जो एक या ज्यादा specific कार्य करता है। यह बाकी programming languages के procedure की तरह ही होता है।

Procedure का एक Header होता है और एक 'body' होती है।

- **Header:** Header में procedure का नाम होता है और procedure को दिए जाने वाले vairiable के मापदंड (parameters) होते हैं।
- Body: Body में एक declaration section होता है, एक execution section होता है, और एक exception section होता है। किसी भी सामान्य PL/SQL Block की तरह है।

Procedure को मापदंड (Parameter) कैसे दिए जाए?

जब भी हम कोई procedure बनाते हैं, हमें उसके लिए कुछ मापदंड define करने पड़ते हैं। Procedure में मापदंडों (parameters) को pass करने के तीन तरीके हैं।

1. **IN parameters:** IN Parameters को Procedures या function के द्वारा Refer किया जाता है। Procedure या function parameters की value को नहीं बदल सकते।
2. **OUT parameters:** OUT parameters को porocedure या Functions द्वारा Refer नहीं किया जा सकता, परन्तु Parameters की values को procedure या function के द्वारा बदला जा सकता है।
3. **IN OUT parameters:** IN OUT Parameters को procedure या function के द्वारा Refer भी किया जा सकता है और ये उसकी value को बदल (overwrite) भी सकते हैं।

PL/SQL Create Procedure

Syntax:

```
CREATE [OR REPLACE] PROCEDURE procedure_name
    [ (parameter [,parameter]) ]
IS
    [declaration_section]
BEGIN
    executable_section
[EXCEPTION
    exception_section]
END [procedure_name];
```

उदाहरण:

इस उदाहरण में, हम user table में record insert कर रहे हैं। इसलिए हमें पहले user table बनानी पड़ेगी।

Table Creation (table बनाना)

Create table user(id number(10) primary key,name varchar2(100));

अब record को user table में डालने के लिए procedure लिखिए।

Procedure Code

```
create or replace procedure «INSERTUSER»
(id IN NUMBER,
name IN VARCHAR2)
is
begin
insert into user values(id,name);
end;
/
```

Output:

Procedure created.

Procedure को call करने के लिए PL/SQL Program

ऊपर बनाए गए procedure को call करने (इस्तेमाल करने) के लिए नीचे code दिया गया है–

```
BEGIN;
   insertuser(120,'Ram');
   dbms_output.put_line('record inserted successfully');
END;
/
```

ID	*Name*
120	Ram

अब USER table को देखिए, उसमें एक record insert हो गया है।

PL/SQL का Drop Procedure

Syntax:

DROP PROCEDURE procedure_name;

उदाहरण:

DROP PROCEDURE pro1;

6.7 TRANSACTION

एक Database transaction काम की atomic unit है, जिसमें एक या एक ज्यादा संबंधित SQL Statements होती है। इसे atomic unit इसलिए कहते हैं क्योंकि Database के अंदर SQL Statements से जो Modification (बदलाव) होते हैं, उनमें transaction को पूरा (committed) किया जाता है, मतलब या तो database में वो change permanenet (स्थायी) हो जाता है या database से पूरा Roll back वापस हो जाता है।

एक सफलतापूर्वक execute (कार्यान्वित) हुई SQL statement और एक committed (पूरी) transaction एक समान नहीं होती। अगर एक SQL statement सफलतापूर्वक execute हो जाए, तब भी, जब तक उसकी transaction committed ना हो जाए, उसे वापस किया जा सकता है और statement द्वारा किए गए सभी बदलावों को भी वापिस किया जा सकता है।

किसी Transaction को शुरू या खत्म करना।

एक transaction की एक शुरूआत (Begining) और एक अंत (END) होता है। एक transaction शुरू होती है जब इनमें से कोई एक घटना घटती है

- किसी SQL Statement को perform किया जाता है, database से connect करने के बाद।
- Transaction के complete होने के बाद नई SQL statement को issue किया जाता है।

एक transaction खत्म (end) हो जाएगी, जब नीचे दिए हुए events (घटना) में से कोई event होगा–

- एक COMMIT या ROLLBACK Statement का इस्तेमाल किया जाए।
- एक DDL Statement जैसे CREATE TABLE को इस्तेमाल (या issue) किया जाए, इस case में COMMIT अपने आप perform हो जाती है।
- एक DCL Statement जैसे GRANT Statement को issue किया जाए, क्योंकि इस case में COMMIT अपने आप (automatically) perform हो जाता है।
- User database से disconnect (अलग) हो जाए।
- User SQL*PLUS से बाहर (exit) हो जाए exit Command का इस्तेमाल करके, यहाँ COMMIT अपने-आप पूरा हो जाएगा।
- SQL*PLUS असमान्य रूप से बंद हो जाए, यहाँ ROLLBACK अपने आप (automatically) perform हो जाएगा।
- कोई DML Statement fail हो जाए, इस case में भी ROLLBACK अपने-आप perform हो जाएगा और उस DML statement को undo (वापस) कर देगा।

किसी Transaction को Commit करना

किसी transaction को SQL की COMMIT COMMAND का इस्तेमाल करके स्थायी बनाया जा सकता है। COMMIT command का सामान्य Syntax है–

```
COMMIT;
```

उदाहरण के लिए:

```
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (1, 'Ram', 32, 'Ahmedabad', 2000.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (2, 'Lalit', 25, 'Delhi', 1500.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (3, 'Sharma', 23, 'Kota', 2000.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (4, 'Chaitali', 25, 'Mumbai', 6500.00 );

COMMIT;
```

Transactions को Roll Back (वापस) करना

Database में किए बदलावों को (अगर COMMIT नहीं हुआ है तो) ROLLBACK Command से वापस किया जा सकता है।

ROLLBACK Command का सामान्य Syntax है–

```
ROLLBACK [TO SAVEPOINT < savepoint_name>];
```

जब कोई transaction किसी कारणवश बीच में बंद हो जाए, जैसे system के fail हो जाने से, तो पूरी transaction को वापस लिया जा सकता है। (जब तक COMMIT न हुआ हो)। अगर आप SAVEpoint का इस्तेमाल नहीं करते हैं, तो नीचे दी गई statement को इस्तेमाल करे changes को ROLLBACK करने के लिए–

```
ROLLBACK;
```

Savepoints

Savepoints एक तरह के Markers होते हैं जो एक बड़ी transaction को छोटी इकाइयों में तोड़ देते हैं, कुछ checkpoints को set (स्थापित) करके। Savepoints को set करके एक बड़ी transaction के अंदर (अगर जरूरत है तो) एक checkpoint तक ही ROLLBACK किया जा सकता है। इसे SAVEPOINT Command के द्वारा किया जा सकता है।

Savepoint का सामान्य syntax है:

```
SAVEPOINT < savepoint_name >;
```

उदाहरण के लिए:

```
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (7, 'Ram', 27, 'HP', 9500.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (8, 'Riya', 21, 'WB', 4500.00 );
```

```
SAVEPOINT sav1;
UPDATE CUSTOMERS
SET SALARY = SALARY + 1000;
ROLLBACK TO sav1;

UPDATE CUSTOMERS
SET SALARY = SALARY + 1000
WHERE ID = 7;
UPDATE CUSTOMERS
SET SALARY = SALARY + 1000
WHERE ID = 8;

COMMIT;
```

ROLLBACK TO sav1: यह statement एक point तक किए गए सभी बदलावों (charges) को वापिस कर देती है, यहाँ savepoint को sav1 के रूप में Mark किया गया है। इसके बाद जो अपने नए बदलाव किए हैं वो शुरू हो जाएँगे।

Automatic Transaction Control

जब भी हम INSERT, UPDATE या DELETE Command को execute करते हैं, तो COMMIT को automatically (अपने-आप) execute करने के लिए, हम AUTOCOMMIT का इस्तेमाल कर सकते हैं:

```
SET AUTOCOMMIT ON;
```

आप Auto commit को नीचे दी गई command से बंद भी कर सकते है–

```
SET AUTOCOMMIT OFF;
```

6.8 TRIGGERS

Triggers को Oracle द्वारा automaticaly बुलाया जाता है (लागू किया जाता है), जब भी कोई विशेष event (घटना) होती है। Triggers को Database में store किया जा जाता है, और बार-बार बुलाया जा सकता है, जब भी कोई विशेष शर्त पूरी होती है।

Triggers stored programs होते है, जो अपने आप execute या fired हो जाते हैं, जब कोई event (घटना) होता है।

Triggers को नीचे दिए गए events के होने पर respond करने के लिए (चलने के लिए) लिखा गया है–

- एक Database Manipulation (DML) Statement जैसे कि (DELETE, INSERT और UPDATE)
- एक Database definition statement (DDL) जैसे (CREATE, ALTER या DROP)
- Database का कोई operation जैसे (SERVERERROR, LOGON, LOGOFF, STARTUP या SHUTDOWN)

Triggers को table, view, Schema या database कही पर भी लगाया जा सकता है। जिसके साथ event जुड़ा हुआ हो।

Triggers के फायदे

Triggers को निम्नलिखित फायदे हैं:

- Triggers कुछ columns की values को automatically generate कर देते हैं।
- Refrential Integrity को लागू करते हैं।
- Table को access करने पर Information store करते हैं।
- Auditing के काम आते हैं।
- Table के Replication को synchronize (तालमेल/बिठाना) करते हैं।
- Invalid transactions (अवैध गतिविधियाँ) को से बचाते हैं।
- सुरक्षा के हिसाब से authorization लागू करते है।

Trigger को Create करना

Trigger Create* करने के लिए *Syntax:

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
   Declaration-statements
BEGIN
   Executable-statements
EXCEPTION
   Exception-handling-statements
END;
```

Here,

यहाँ कौन क्या काम कर रहा है वह देखिए–

- **CREATE [OR REPLACE] TRIGGER trigger_name:** यह trigger के नाम के साथ कोई नया trigger बनाता (create) है या किसी मौजूदा trigger को नए से बदल देता है।
- **{BEFORE |AFTER|INSTEAD OF}:** यह बताता है कि trigger कब काम करना शुरू करेगा। INSTEAD of CLAUSE को किसी VIEW में trigger create करने के लिए इस्तेमाल किया जाता है।
- **{INSERT [OR] | UPDATE [OR] | DELETE}:** यह DML operations को specify करता है।
- **[OF col_name]:** यह उस column का नाम बताता है जिसको update करना है।
- **[ON table_name]:** यह उस table का नाम specify करता है (बताता है) जिस पर trigger लगा हुआ है।

- **[REFERENCING OLD AS o NEW AS n]:** यह विभिन्न DML Statements जैसे– INSERT, UPDATE और DELETE, के लिए नई और पुरानी values को refer करता है।
- **[FOR EACH ROW]:** यह एक ROW level trigger को specify करता है, जिसका मतलब है कि trigger प्रत्येक row, जिस पर प्रभाव पड़ता है, के लिए execute हो। वरना trigger केवल एक बार execute होगा जब SQL Statement execute होगी, जिसे table level trigger कहते हैं।
- **WHEN (condition):** यह जिन ROWS पर trigger को चलना है उनके लिए शर्त (condition) बताता है। यह केवल ROW level के triggers के लिए वैध है।

6.9 CURSORS

SQL Statements को execute करने के लिए, Oracle engine द्वारा एक work area (कार्य करने का क्षेत्र) इस्तेमाल किया जाता है, जो उसकी Internal processing और Information को store करने के काम आता है।

यह SQL Operations का निजी area होता है। PL/SQL के 'CURSOR' से हम इस area को नाम दे सकते हैं, और उसमें stored Information को access कर सकते हैं।

Cursor के इस्तेमाल

Cursor का मुख्य कार्य है Data को Retrieve करना (पुन: प्राप्त करना), यह Result Set में से एक बार में एक ROW उठाता है, ना कि SQL Command की तरह जो एक Result Set (table) की सभी पर rows पर एक साथ काम करता है। cursors का इस्तेमाल तब किया जाता है जब user को records को एक एक करके या Row by Row update करना होता है, किसी Database table में।

जो Data cursor में stored होता है उसे Active Data Set कहते हैं। Oracle DBMS में एक और predefined area होता है जो Main Memory में होता है जिसके अंदर cursors open (खुलते) होते हैं। इसलिए cursor का size इस area के अनुसार सीमित होता है।

Cursor के Actions (कार्य)

- **Declare Cursor:** किसी cursor को declare करने के लिए SQL statement को define (परिभाषित) करना पड़ता है जो कोई Result set (table) देती है।
- **Open:** एक cursor उसके द्वारा define की गई SQL statement को execute करने पर open होता है।
- **Fetch:** जब cursor open (खुल) हो जाता है, तब rows को एक एक करके या किसी block में cursor से प्राप्त (fetch) किया जा सकता है data manipulation को पूरा करने के लिए।
- **Close:** Data manipulation के बाद cursor को स्पष्ट रूप से बंद कर देता है।
- **Deallocate:** अंत में, cursor definitions को delete कर देता है और cursor से जुड़े हुए system के सभी resources को release (छोड़ देता है) कर देता है।

Cursors के प्रकार

Cursors को उनके खुलने की परिस्थितियों के हिसाब से बाँटा जाता है–

- **Implicit Cursor:** अगर oracle engine cursor को अपनी Internal processing के लिए खोलता (open करता) है, तो उसे IMplicit cursor कहते हैं। जब भी कोई Query execute होती है यह user के लिए oracle द्वारा अपने आप create कर दिया जाता है और इसकी coding सरल होती है।
- **Explicit Cursor:** जब cursor को PL/SQL block के द्वारा data-processing के लिए किसी की मांग पर (on-demand) खोला जाता है तो ऐसे user-defined cursor को explicit cursor कहते हैं।

Explicit Cursor

Explicity cursor को PL/SQL के declaration section में define किया जाता है। इसे एक SELECT Statement पर create किया जाता है जो एक से ज्यादा row return करती है।

Syntax:

```
CURSOR cursor_name IS select_statement;
```

जहाँ

`cursor_name` - cursor के लिए एक उपयुक्त नाम है।

`select_statement` - एक select query है जो विभिन्न row return करती है।

Explicit Cursor को कैसे इस्तेमाल किया जाए?

एक Explicit cursor को इस्तेमाल करने के चार Step हैं–

1. Declaration Section में Cursor को Declare करें।
2. Execution Section में Cursor को open करें।
3. Execution Section में Cursor से Data Fetch (निकालें) करें।
4. Cursor को execution section में बंद (close) कर दें, PL/SQL block को खत्म करने से पहले।

Syntax:

```
DECLARE variables;
 records;
 create a cursor;
 BEGIN
OPEN cursor;
FETCH cursor;
 process the records;
 CLOSE cursor;
END;
```

6.10 GRANTING AND REVOKING

(Special Permission को Grant करना या उनको Revoke (वापस लेना) करना)

Data control Langauge (DCL) को database में विशेषाधिकारों (Privilege) को नियंत्रण करने के लिए इस्तेमाल किया जाता है। Database में कोई भी operation perform करने के लिए जैसे table create करना, sequence या view create करना, आदि के लिए नेमत को विशेषाधिकार चाहिए होते हैं। विशेषाधिकार दो तरह के होते हैं।

- **Systems के विशेषाधिकार:** इनके अंदर किसी session, table, create करने और system के सभी तरह के विशेषाधिकार शामिल होते हैं।
- **Object के विशेषाधिकार:** इसमें Database table में कोई operation करने के लिए, किसी command या Query की अनुमित के विशेषाधिकार शामिल होते हैं।

DCL में दो तरह की commands होती हैं–

- **GRANT:** User को access के या अन्य प्रकार के विशेषाधिकार देने के लिए इस्तेमाल की जाती है।
- **REVOKE:** User से विशेषाधिकार वापस लेने के लिए इस्तेमाल की जाती है।

User को Session Create करने की अनुमति देना

जब हम SQL में कोई user create करते हैं, बिना उचित अनुमति या विशेषाधिकार के वह कुछ नहीं कर सकता यहाँ तक की login भी नहीं कर सकता और session भी create नहीं कर सकता। User को session create करने के विशेषाधिकार देने के लिए नीचे दी गई command इस्तेमाल की जाती है।

```
GRANT CREATE SESSION TO username;
```

User को Table Create करने की अनुमति देना

User को table create करने की अनुमति देने के लिए नीचे दी गई command इस्तेमाल की जाती है–

```
GRANT CREATE TABLE TO username;
```

यह नेमत को table space में table store करने का space प्रदान करती है।

User को Create table की अनुमति देना, उस table में data store करने के लिए पर्याप्त नहीं है। हमें user को उपलब्ध table space को उसके table और data के लिए इस्तेमाल करने का विशेषाधिकार भी देना पड़ेगा।

```
ALTER USER username QUOTA UNLIMITED ON SYSTEM;
```

यह command user को unlimited table space इस्तेमाल करने की अनुमति देती है।

Permissions (विशेषाधिकारों) को वापस लेना

अगर आप किसी user से विशेषाधिकारों का वापस लेना चाहते हैं तो REVOKE COMMAND का इस्तेमाल होता है-

```
REVOKE CREATE TABLE FROM username
```

अभ्यास प्रश्न

1. PL/SQL क्या है?
2. PL/SQL के फायदे बताइए।
3. PL/SQL के operators के बारे में बताइए।
4. PL/SQL variables के बारे में बताइए।
5. PL/SQL के user defined functions के बारे में बताइए।
6. Cursors क्या होते हैं?
7. PL/SQL में Flow control को विस्तार से समझाइए।
8. PL/SQL के FOR LOOP को उदाहरण के साथ समझाइए।
9. किसी number का factorial निकालने के लिए PL/SQL में program लिखिए।
10. Procedure क्या होता है?
11. Triggers क्या होते हैं?
12. Translations क्या होती है?
13. Triggers के प्रकारों को विस्तार में बताइए।
14. RETURN Statement का महत्त्व क्या है?
15. PL/SQL की विशेषताएँ बताइए।
16. हम Trigger को कैसे Drop कर सकते हैं।
17. PL/SQL Block के तीन basic section (हिस्से) कौन से हैं?
18. Cursor के attribute SQL% NOT FOUND से क्या Return होगा?
19. Triggers के uses क्या हैं?
20. Trigger में WHEN Clausre का क्या use है?

❑❑❑

7

NO-SQL

7.1 NO-SQL: परिचय (INTRODUCTION)

NOSQL एक non-relational database management system है जो पारंपरिक relational database से कई महत्त्वपूर्ण तरीकों से अलग है। यह उन distributed data stores के लिए design किया गया है जहाँ बहुत ज्यादा data storing की जरूरत पड़ती है। (जैसे, Google, Facebook जो प्रतिदिन users का Data store करते है, वो भी Terabites में) इस तरह के Data Storage में Fixed Scheme की जरूरत नहीं होती, और यहाँ Join Operation को भी avoid किया जाता है और आमतौर पर data को horizontally मापा जाता है।

RDBMS से NoSQL की तुलना

RDBMS

- इसमें Structured (संरचित) और Organised (संगठित) Data होता है।
- यह Structured Query Language (SQL) का इस्तेमाल करता है।
- Data और उसके Relationships अलग-अलग Table में Store होते हैं।
- यह Data Manipulation Language (DMC) और Data Definition Language (DDL) का प्रयोग करता है।
- इसमें कड़ी स्थिरता होती है। (High Consistency)

NoSQL

- इसका मतलब है Not Only SQL
- इसकी कोई घोषित Query Language नहीं हैं।
- कोई पहले से Defined की हुई Scheme नहीं होती।
- यह Key-value pair storage, Column store, Document store और Graph Database का इस्तेमाल करता है।
- इनमें Eventual Consistency (स्थिति के अनुरूप स्थिरता) होती है।
- Data Unstructured और Unpredictable (अप्रत्याशित, जिसका अनुमान ना लगाया जा सके) होता है।
- CAP theorem का इस्तेमाल होता है।

- इस तरह के database में बेहतर (Performance) प्रदर्शन बेहतर availability और बड़े पैमाने पर data को handle करने पर ज्यादा ध्यान दिया जाता है।
- BASE transaction का उपयोग किया जाता है।

NoSQL के फायदे

NoSOL database जैसे कि MongoDB. और Cassandra के साथ काम करने के बहुत सारे फायदे होते है। सबसे बड़ा फायदा है : High scalability (data को बड़े पैमाने पर कर पाना) और High availability (data का हर वक्त और हर रूप में available होना)

1 **High scalability:** NoSQL data base data की horizontal scaling के लिए sharding का इस्तेमाल करते है। Data को अलग-अलग करना और उसे बहुत सारी machine (Computer System) पर इस तरह रखना की data का क्रम वही रहें; इस प्रक्रिया को sharding कहते है। Vertical scaling का मतलब होता है, कि एक ही machine पर data को handle करने के लिए और साधन दे दिए जाए, और horizontal scaling का मतलब है कि data को handle करने के लिए और machine दे दी जाए। Vertical scaling को लागू करना आसान नहीं होता परन्तु horizontal scaling को लागू करना आसान होता है। Horizontal scaling के उदाहरण है: MongoDB, Cassandra इत्यादि। NoSQL scalability की वजह से बहुत ज्यादा data को संभाल सकते है, जैसे data बढ़ता जाता है। NoSQL database में data को संभालने के लिए अपना पैमाना बढ़ा लेता है।

2. **High availability:** NoSQL database auto replication (अपनी प्रतिकृति अपने आप बना लेना) feature होता है जो किसी भी failure की स्थिति में data की एक copy या कृति बना लेता है, जिससे data की उपलब्धता बढ़ जाती है। यह data failure होने पर, आखिरी बार जब data स्थिर था उस स्थिति को वापस ला देता है और पूरे data को नष्ट नहीं होते देता।

NoSQL के नुकसान (disadvantages)

NoSQL के कुछ disadvantages नीचे दिए गए हैं।

1. **Narrow focus:** NoSQL database को मुख्यत storage के लिए design किया गया है इसलिए यह functionality (कार्यप्रणाली) पर बहुत ज्यादा focus (ध्यान) नहीं करता। जहाँ पर बहुत ज्यादा transaction को Manage करना हो, वह RDBMS एक बेहतर विकल्प होता है।
2. **Open-source:** NoSQL एक open-source database है। इसके लिए कोई भरोसे मंद standard नहीं है।
3. **Management challenge:** बहुतसारे data को mange करने के लिए big data tools का इस्तेमाल करना आसान हो सकता है परन्तु सभी चीजों को manage करना उतना भी आसान नहीं है। NoSQL में data प्रबंधन RDBMS से ज्यादा जटिल है। NoSQL को install करना और manage करना बहुत चुनौतीपूर्ण और थकाने वाला काम है।
4. **GUI:** (Graphical use interface-user से Graphical Method से जुड़ना) की सुविधा उपलब्ध नहीं है।
5. **Backup:** Data का backup कुछ NoSQL databases जैसे कि MongoDB की कमजोरी है। MongoDB के पास data का स्थिर तरीके से backup लेने की सुविधा नहीं है।

6. **बहुत बड़ा Document Size:** NoSQL database जैसे MongoDB या CouchDB data को JSON format में store करते हैं। इसका मतलब है कि Document बहुत बड़े होते है। (Big data, network bandwidth, speed) और descriptive key names होना बहुत परेशान करता है, क्योंकि size बड़ा होता है।

NoSQL Database के प्रकार

1. MongoDB database NoSQL के document based database में आता है।
2. **Key value store के प्रकार:** Memcached, Redis, Coherence.
3. **Tabular के प्रकार:** Hbase, Big Table, Accumulo.
4. **Document based:** MongoDB, CouchDB, Cloudant.

NoSQL को कब इस्तेमाल करना चाहिए

1. जब बहुत ज्यादा बड़ा data store करना हो।
2. Data के बीच की relationship का बहुत ज्यादा महत्व ना हो।
3. Data समय के साथ बदलता रहता हो और structured. ना हो।
4. Database level पर Constraints और Joints की आवश्यकता ना हो।
5. Data लगातार बढ़ता जाए और उसे संभालने के लिए database के पैमाने को लगातार बढ़ाना पड़े।

NoSQL Databases के प्रकार

NoSQL databases की मुख्यत: चार श्रेणियाँ होती है। प्रत्येक श्रेणी. (category) की अपनी-अपनी विशेषता और सीमाए है। कोई भी एक तरह का database सभी problems को solve नहीं कर सकता। आपको अपनी जरूरत के हिसाब से database चुनना पड़ेगा।

- Key-value Pair पर आधारित database
- Column-oriented Graph पर आधारित
- Graphs पर आधारित
- Document-oriented (Documents पर केन्द्रित database)

Key-Value Pair पर आधारित

Data को key/value pairs में Store किया जाता है। इसे बहुत सारे data और heavy load को handle करने के लिए design किया गया है। Key-value pair database data को hash (#) table के रूप में store करते है, जहाँ प्रत्येक key अलग (unique) होती है, और value JSON, BLOB (Binary Large Objects), string, आदि के रूप में हो सकती हैं.

Column-based पर आधारित

Column पर केन्द्रित databases columns पर काम करते है और Google द्वारा बताए गए Big Table Paper पर आधारित होते हैं। हर column को अलग से treat किया जाता है।

यह aggregation queries जैसे : SUM, COUNT, AVG, MIN आदि के साथ बहुत अच्छा प्रदर्शन करता है, क्योंकि सारा data column में होता है।

Column पर आधारित NoSQL databases को अधिकतर data warehouses, business intelligence, CRM, Library card catalogs आदि का data mange करने के लिए इस्तेमाल किया जाता है।

HBase, Cassandra, Hypertable column based database के कुछ उदाहरण है।

Document-Oriented (Documents पर केन्द्रित)

Document पर केंद्रित NoSQL database data को key value pair की तरह ही store करते है। परन्तु value को document के रूप में stored किया जाता है। Document को JSON या XML के formats में store किया जाता है। Database value को समझ सकता है और उस पर query इस्तेमाल की जा सकती है।

इस तरह के database को अधिकतर (MS systems, blogging platforms, real-time analytics और e-commerce में इस्तेमाल किया जाता है।

इसे बहुत जटिल transactions में इस्तेमाल नहीं करना चाहिए जहाँ बहुत operations और queries का इस्तेमाल होता हो।

Graphs पर आधारित

यह data को entities के रूप मे store करता है और entities के बीच के Relation को भी store करता है। जहाँ entities बीच (node पर) में होती है और relationship किनारों (edges पर) होती है। Edge (दो entitie/nodes) के बीच के संबंध को बताती है। प्रत्येक edge और node का एक unique identifier होता है।

Relational database से तुलना करने पर जहाँ tables इतनी अच्छी तरह से connect नहीं होती, एक Graph database multi-relational (अनेक प्रकार की relationship) वाला होता है। Traversing relationship भी बहुत fast होती है, क्योंकि यह पहले ही database में होती और उन्हें calculate करने की भी जरूरत नहीं पड़ती।

Graphs पर आधारित database को अधिकतर social networks, logistics, spatial data के लिए इस्तेमाल किया जाता है। Neo4J, Infinite Graph, OrientDB, FlockDB कुछ popular graph आधारित database है।

7.2 USAGES (उपयोग)

Industry में NoSQL का उपयोग

1. Session को Store करने के लिए

- Relational database का इस्तेमाल करके session data को manage करना बहुत मुश्किल है, खासकर जब applications बहुत ज्यादा हों।
- इस तरह के cases में सही तरीका है कि global session store का इस्तेमाल करना, जो site पर आने वाले प्रत्येक user की session information को manage करता है।
- NoSQL database जो web application size में बहुत बड़ी होती है, उनको store करने के लिए उपयुक्त होता है।

- चूंकि session का data unstructured रूप में होता है, तो बिना schema वाले documents में उसे store करना आसान होता है, ना कि relation database के record में।

2. User के Profile को Store करने के लिए

- Online transactions को पूरा करने के लिए user की पसंद-नापसंद, user का authentication (प्रमाण) करने के लिए user के profile (जानकारी) को store करना जरूरी है।
- हाल ही में, web और mobile application का प्रयोग बहुत ज्यादा बढ़ गया है। Relational database इतने बड़े पैमाने पर user की जानकारी को (profile) को नहीं संभाल सकता, क्योंकि इसमें केवल एक server होता है।
- NOSQL में server की संख्या को बढ़ाकर capacity (Data store करने की क्षमता) को बढ़ाया जा सकता है।

3. Content और Metadata को Store करना

- बहुत सी companies जैसे कि publication houses को ऐसी जगह चाहिए जहाँ वो बहुत ज्यादा data को store कर सकें जैसे articles, digital content और e-books, ताकि सभी tools को मिलाकर एक learning platform बनाया जा सकें।
- वो applications जो content पर आधारित होती है, उनमें metadata को बार-बार access करना पड़ता है, जिसमें response times बहुत कम होना चाहिए।
- Content पर आधारित applications को बनाने के लिए, NoSQL का प्रयोग data को जल्दी access करने के लिए और विभिन्न प्रकार से content को store करने के लिए किया जाता है।

4. Mobile Applications में प्रयोग

- चूंकि smartphone के users बहुत ज्यादा बढ़ गए है, mobile applications को बढ़ते हुए data volume की समस्या का सामना करना पड़ रहा है।
- NoSQL का प्रयोग करके, mobile application कम size के साथ शुरू होता है और जब user की संख्या बढ़ जाती है, तो application की क्षमता को भी आसानी से बढ़ाया जा सकता है, जो relational databases में बहुत मुश्किल है।
- NoSQL database में data को बिना schema (structure) के store किया जाता है, इसलिए application developer के लिए apps को database में बहुत ज्यादा बदलाव किए बिना update करना बहुत आसान होता है।
- Mobile Companies जैसे Kobo और Playtika, NOSQL का इस्तेमाल करती है और लाखो users को सेवाएँ दे रही है।

5. Third-Party के data को Aggregation (एकत्र) करते के लिए

- किसी business में third party का data access करने की जरूरत बार-बार पड़ती है। जैसे कि कोई consumer goods बनाने वाली कंपनी को store से की गई बिक्री (sale) का data भी चाहिए होता है, और दूकानदार की खरीददारी का data भी चाहिए होता है।
- इस तरह के scenarios में, NoSQL databases बिलकुल उपयुक्त होता है।

6. Internet of Things (वस्तुओं का Internet)

- आजकल, करोड़ो उपकरण devices internet से जुड़े हुए है, जैसे smartphones, tablets घर के उपकरण hospitals के उपकरण, गाड़ियाँ warehoused इत्यादि। इस तरह के उपकरणों से बहुत सारा और बहुत तरह का data generate (उत्पन्न) होता है और होता जा रहा है।
- Relational databases इस तरह के data को store नहीं कर सकता। NOSQL database, organizations (संगठनों) को इन करोड़ो उपकरणों से निकलने वाले data को access करने के लिए, store करने के लिए जरूरी क्षमता का विस्तार करने की सुविधा प्रदान करता है।

7. E-Commerce में उपयोग

- E-commerce कंपनियाँ NoSQL का इस्तेमाल बहुत बड़े data को store करने के लिए और user द्वारा किए गए बहुत सारे अनुरोधों को store करने के लिए करती हैं।

8. Social Gaming में उपयोग

- बहुत सारी applications जैसे social games जो गहन data पर आधारित है, उनके users लाखों तक बढ़ गए है। Users की संख्या का इस तरह बढ़ना और बहुत ज्यादा data, ऐसे database की जरूरत बढ़ाता है, जो इस तरह के data को store कर सके, NOSQL इस तरह के applications के लिए उपयुक्त होता है।
- NOSQL बहुत mobile gaming कंपनियो जैसे: electronic arts, zynga और tencent द्वारा इस्तेमाल किया जाता है।

9. Ad Targeting में उपयोग

- किसी current web page पर ads या offers को display करना, income से जुड़ा हुआ फैसला होता है। web page में ads को कहाँ पर display करना है, किस user के group को target करना है; या user के behavioral और demographic characteristics की जानकारियाँ एकत्र करने के लिए NoSQL platform का इस्तेमाल किया जाता है।
- NoSQL database ad कंपनियों को user की details track करने की सुविधा प्रदान करते है, और बहुत quickly detail को place करके clicks की संभावनाओं को बढ़ा देता है।
- AOL, Mediamind और PayPal कुछ ad कंपनियाँ है, जो NoSQL database का इस्तेमाल करती है।

7.3 APPLICATION (प्रयोग)

अक्सर लोग कोई particular platform खरीद लेते है, उस पर चलने वाले apps के कारण बहुत से NoSQL पर आधारित applications app की category में आते हैं। ये applications नहीं बन पाती अगर ये relational database का इस्तेमाल करती।

Facebook messaging platform

Apache Cassandra database को Facebook के द्वारा बनाया गया था, उनके inbox को powerful बनाने के लिए। बहुत सालों तक उसने यह काम किया। Cassandra नीचे दिए गए तरीके से काम करता है:

- Cassandra users के messages और terms (शब्द आदि) को Index करता है और messages को content के हिसाब से ढूंढ़ने (search) करने की सुविधा देता है। User ID यहाँ primary key है। प्रत्येक term super column और message ID, column का नाम होता है।
- Cassandra किसी विशेष user को भेजे गए या उससे आए हुए messages (संदेशो) की list बनाने की सुविधा प्रदान करता हैं। यहाँ पर user ID primary key, recipient (message receive करने वाला) की ID super columns और message ID column names होते है।

Facebook में original Cassandra paper में सभी recent information (नई जानकारी) होती है, और इसे DataStax के द्वारा maintain किया जाता है, जो Cassandra को promote करने वाली बड़ी कंपनी है।

Amazon DynamoDB

Amazon ने मौलिक रूप से Dynamo paper को प्रकाशित (Publish) किया, जिसका उद्देश्य NoSQL के key-value stores पर आधारित database को launch करना था। उसके बाद से Amazon ने एक अलग database बनाया जिसका नाम था DynamoDB जो Amazon की marketplace site के Web Service में उपलब्ध था।

हालांकि Dynamo को अपना नाम original Dynamo से मिला है, इसकी approach उससे अलग है: DynamoDB worldwide synchronous replication (पूरे विश्व में हर जगह एक जैसी ही कृति या copy) प्रदान करता है जिससे consistency और durability (टिकाऊ होने) की गारंटी मिलती है जो किसी भी enterprise application के लिए बहुत जरूरी है।

DynamoDB में आपको प्रति घंटा इस्तेमाल करने के हिसाब से कीमत चुकानी पड़ती है ना कि कितना data store किया है उसके हिसाब से, जो की एक अलग और दिलचस्प model था और application developers द्वारा बहुत पसंद किया गया।

इसमें एक 'free tier' का विकल्प भी होता है, जिसमें 25GB का data store करने की सुविधा और कुछ write और read capacity units दी जाती है।

Google Mail

Google की Bigtable को google की applications को एक wide-column storage देने के लिए बनाया गया था, जैसे Orkut, Google Earth, Google Maps, Google Books, You tube, blogger. com, Google Code और Google Mail.

Bigtable बहुत large information के set के लिए index lookup tables provide करती है।

LinkedIn

Linkedin ने information (जानकारियों) के मंथन के लिए Hadoop का इस्तेमाल किया था, जो एक रात में ही relationship की जानकारियों का मंथन (उन पर processing) करके उसे graph information के रूप में NoSQL के Voldemort key-value store को अगले दिन की query के लिए दे सके।

इस तरह Linkedin ने service के सारे data के rolling view (एक चक्र में) को maintain किया हुआ था।

BBC का iPlayer online media catalog

British Broadcasting Corporation (BBC) के पास अपने UK नागरिकों के लिए ऐसी online सुविधा है जिसे iPlayer कहते है, इसके द्वारा वह BBC के television और radio के कार्यक्रमों को मुफ्त में देख सकते हैं।

Episodes, series, और brands की जानकारियों को updated करने की जिम्मेदारी एक अलग team की होती है।

BBC अब अनेक MySQL systems की जगह एक single (अकेले) Mark Logic Server 6 का उपयोग कर रही है, जो program metadata का access प्रदान करता है। इस प्रक्रिया में एक data services API जिसका नाम Nitro था, को बनाया गया और उसे Mark Logic Server के अंदर डाल दिया गया।

Nitro बढ़ती हुई BBC की services को ताकत प्रदान करता है। Nitro की शुरूआत iPlayer की functionality (कार्यक्षमता) में बदलाव के साथ हुई जिससे इस platform की स्थिरता बढ़ती है। भविष्य में, Nitro में partner organizations को भी information देने की सुविधा होगी।

BBC Sport और Olympics platforms

सन् 2011 में, BBC ने महसूस किया कि journalists (पत्रकार) यह निर्णय लेने में बहुत समय बर्बाद करते है कि BBC की Sport website पर कहानियों को कहाँ Publish (प्रकाशित) करता है। यह बहुत ज्यादा वक्त और पैसा बर्बाद करता था, और user के लिए कहानिया लगातार sports website पर उपलब्ध नहीं होती थी।

इसलिए BBC ने इस प्रक्रिया को automatic बनाने के लिएबिलकुल अलग तरह का solution निकाला जिसे Dynamic Semantic Publishing (DSP) कहा जाता है। Mark Logic Server 6 और Ontotext's GraphDB (पहले जिसका नाम Big OWLIM था) को एक साथ इस्तेमाल करने से, BBC को पत्रकारो को कहानियों के लिए विषय सुझाने की सुविधा मिल गई।

इस नए तरीके से BBC को कहानियों के Subject से यह निधार्रित करने की भी सुविधा मिली की कहानियों का कहाँ प्रकाशित किया जाए, इससे प्रत्रकारों पर उसकी निर्भरता कम हो गई।

BBC Sport के home page पर जाकर England football team के link पर click करके, ना केवल आप England football team की कहानियाँ पढ़ सकते है, बल्कि England football team के किसी player के बारे में जानकारी, उसके परिवार की कहानियाँ भी पढ़ सकते है।

Health Care.gov

Health care.gov को अब तक का सबसे जटिल IT system कहा जाता है। इसको बनाने के बहुत सारे system की आवश्यकता पड़ी, जिन्हें Health Care. gov marketplace (जो screen पर दिखाई देता है) के साथ जोड़ा गया।

Scenes के पीछे, बहुत सारे और systems होते है जो सहायक के रूप में कार्य करते है, जिसमें अलग-अलग एजेंसिज से information (जानकारियाँ) store करना शामिल है, जैसे कि Insurance का data और जानकारियाँ जो states अपने नागरिको को प्रदान करते है। इसके अलावा, इस पर Insurers उन policies को भी marketplace website पर डाल सकते है, जो वह नागरिको को देना चाहते है।

कई तरह के systems के बीच के communication के लिए messages (संदेशो) को सुरक्षा के लिए store करना जरूरी होता है, ताकि वह बाद में भेजे सकें। हालांकि Health Care. gov 34 states के नागरिको को coverage प्रदान करता है, back-end systems database के द्वारा सभी 50 states के लिए support करता है।

Centers for Medicare या Medicaid Services (CMS) ने Mark Logic को सभी systems data के लिए Back end database के रूप में चुना है। Mark Logic Server इन systems के बीच चलने वाले सभी XML content को store करता है और नागरिको की जरूरतों को पूरा करने की क्षमता प्रदान करता है।

एक subsystem जो सभी Message को real time (वास्तविक समय) में analyzes करता है, NoSQL का अब तक का दिखाई देने वाला (visual) सबसे सफल system सिद्ध हुआ है, जो नागरिको की जिन्दगियों पर सीधा प्रभाव डालता है। हालांकि यह project कुछ परेशानियों को अनुभव करता है, फिर भी इसमें सभी जटिलताओं का handle करके, एक सफल rollout के रूप में सात million Americans को cover किया है।

सुरक्षित Information Sharing

बहुत सी परिस्थितियों में आपको information (सूचना) के access के साथ-साथ उसकी सुरक्षा का भी ध्यान रखना पड़ता है। जैसे कुछ उदाहरण है:

- किताब का प्रकाशक (publisher) किताब खरीदने से पहले उसकी कुछ summary पढ़ने की इज़ाज़त देता है परन्तु पूरी किताब को आप खरीदने के बाद ही पढ़ सकते है।
- कई सामाजिक एजेंसी अलग-अलग लोगों जैसे child protection officers, medical staff, educators और law enforcement agencies को अलग-अलग data access के अधिकार देती है।
- कोई भी intelligence-sharing application जहाँ खोज करके बहुत जरूरी सूचनाएँ shared (साझा) करनी होती है, परन्तु यहाँ अलग-अलग cases (मामलो) के हिसाब से ही access दिया जाता है। जिसको जितनी जरूरत हो उतनी ही information देखने का अधिकार होता है।

यह data की सुरक्षा के अति आवश्यक है। साथ ही, किसी एक ही record के विशेष हिस्से को देखने के लिए हम denormalisation या label based-access control (LBAC) की जरूरत पड़ती है। LBAC record को उसके अंदर रखे गए content के हिसाब से सुरक्षा प्रदान करता है ना कि दी गई permission के हिसाब से, यह label से पता लगा लेता है कि data कितना संवदेनशील है।

इस तरह के मामलों में NoSQL databases record, cell/element के लिए तीन स्तर की सुरक्षा प्रदान करता है, जैसे Accumulo, MarkLogic Server और Allegro Graph कुछ अच्छे विकल्प हैं।

Citizen Engagement (नागरिकों की भागीदारी)

सरकारे NoSQL databases का इस्तेमाल करती है नागरिको के सशक्तिकरण के लिए, ताकि उन्हें पता लगे उनका देश कैसे चल रहा है। उदाहरण के लिए: Virginia, में Fairfax County, जो Mark Logic Server का इस्तेमाल करके online browser के द्वारा, geo spatial information प्रदान कराती है और नागरिको और सरकार के बीच interface को ढूंढती है। यह बहुत तरह की जानकारियाँ उपलब्ध करती है। जैसे: देश के geographices points (भौगोलिक स्थिति) और police से संबंधित कार्यक्रमों की जानकारी।

UK (united kingdom) में एक पुरस्कृत website legislation.gov.uk UK के उन कानून के बारे में भी सारी जानकारी बताती है, जो हजारो साल पहले भी बने हुए थे। अगर आप जानना चाहते हैकि 1542 में Wales में चोरी के लिए क्या कानून था तो आप website से पता कर सकते हैं।

यह वर्तमान में संविधान में क्या नए कानून बनाए है और आने वाले कानूनों की जानकारी भी बताती है। यह नागरिकों के साथ-साथ कानून बनाने वालो के लिए भी बहुत बड़ा reference का स्त्रोत है।

अभ्यास प्रश्न

1. NO-SQL क्या है?
2. NO-SQL के लाभ और हानियाँ बताइये।
3. NO-SQL की RDBMS से तुलना लिखिए।
4. NO-SQL databases के सभी प्रकारों को बताइये।
5. NO-SQL के उपयोग लिखिए।
6. NO-SQL की कोई पाँच application समझाइये।

❑❑❑

8

सुरक्षा (SYSTEM)

8.1 INTRODUCTION (परिचय)

Data एक बहुत जरूरी वस्तु होती है जिसें बहुत ध्यान से और दृढ़ता से संभालने की आवश्यकता होती है जैसे आप किसी आर्थिक संसाधन को संभाल रहे हों। किसी भी Organisation के commercial data के कुछ हिस्से या पूरा data की उसके लिए बहुत महत्ता (importance) होती है, इसलिए Data को सुरक्षित और गुप्त (confidencial) रखना जरूरी होता है। इस Chapter में हम Data की सुरक्षा के बारे में पढ़ेगें। आजकल बहुत तरीके के Computer - based controls (नियंत्रक) आते हैं जो Data को सुरक्षा प्रदान करते हैं।

Data base security वह तकनीक है जो Data की सुरक्षा करती है और Database को Intentional (जानबूझ कर) या आकस्मिक (accidentally) खतरे से बचाती है। सुरक्षा की चिंता सिर्फ उस data तक सीमित नही है जो किसी organisation के database में रहता है, परन्तु सुरक्षा में कमी system के और हिस्सों को भी नुकसान पहुचाँ सकती है, जो आखिर में Database structure को ही प्रभावित करता है। इसलिए data security में hardware, sofrware, human resource (कर्मचारी) और data सभी शमिल होते हैं। security (सुरक्षा) का प्रभावशाली तरीके से उपयोग करने के लिए हमें उचित नियंत्रण चाहिए होते हैं, जिसका काम अलग-अलग हो सकता है परन्तु उद्देश्य system की सुरक्षा होता है। उचित सुरक्षा की जो जरूरत पहले के समय में नकार दी जाती थी, वर्तमान में organisation के द्वारा अनेक तरीको से ध्यान में रखी जाती हैं।

हमे निम्न परिस्थितियों के लिए Database security की आवश्यकता पड़ती है–

- चोरी और धोखाधड़ी
- गोपनीयता और गुप्तता की हानि
- Data की privacy (निजता) की हानि
- Data की अखंडता (Integrity) की हानि
- Data की उपलब्धता की हानि

इन सभी परिस्थितियाँ उन सभी महत्तवपूर्ण areas को बताती है जहाँ पर organisation को ध्यान देना चाहिए, जिससे Data का नुकसान होने का जोखिम कम हो जाए। कुछ परिस्थितियों में अगर किसी एक area में कोई नुकसान होता है तो उसकी वजह से दूसरे area में भी नुकसान हो जाता है, क्योंकि organisation का सारा data आपस में जुड़ा हुआ (connected) होता हैं।

What is a Threat? (खतरा क्या है?)

कोई भी एसी स्थिति या घटना, चाहे वो जानबूझकर हुई हो या अनजाने में, नुकसान कर सकती है, जो Data base के structure पर गलत प्रभाव ड़ालती है और इसके फलस्वरूप oganisation को भी नुकसान होता है। एक खतरा किसी परिस्थिति या घटना जिसमे कोई व्यक्ति शामिल हो से उत्पन्न हो सकता है, जिससे organisation और उसके database को नुकसान पहुँच सकता हो। किसी नुकसान या खतरे के बाद organisation पर किस हद तक प्रभाव पड़ता है यह कई पहलुओ पर निर्भर करता है जैसे कि उसके बचाव के तरीके और आक्स्मिक घटना से निपटने की योजना। उदाहरण के लिए मान लिजिए, कोई hardware खराब हो गया है जिससे secondary storage corrupt हो गई है तो organisation की सभी गतिविधियाँ कुछ समय तक रूक जाएँगी जब तक समस्या ठीक ना हो जाए।

Computer based control (कंप्यूटर पर आधारित नियंत्रण)

किसी computer system में खतरे से बचाव के तरीके बहुत तरह के हो सकते हैं जिनमे physical control से लेकर managerial procedures (प्रबंधन की प्रकिया) शमिल है। पहले से मौजूद बहुत प्रकार के computer based control के अलावा यह ध्यान रखना जरूरी है, की DBMS की सुरक्षा को operating system की सुरक्षा की तरह ही देखना चाहिए क्योंकि ये दोनो आपस में बहुत अच्छी तरह से संवंचित हैं।

अधिकतर computor based data security की list नीचे दी गई है:

- Access authorization (access का अधिकार)
- Access controls (access पर नियंत्रण)
- Views
- Backup and recovery of data (data का Backup - {एक और copy} रखना या Data को दोबारा से Recover करना)
- Data Integrity (Data की अखंड़ता)
- Encryption of Data
- RAID तकनीक

Access Control क्या है?

Database system में access control का मतलब किसी को कुछ अधिकार देना (granting) या अधिकार वापस ले लेना (Revoking) होता है। अधिकार user को database objects को create करने या access (देखने) की सुविधा प्रदान करते हैं, या कुछ विशेष DBMS की utilities को चलाने की सुविधा प्रदान करते हैं। users को अधिकार या सुविधाएँ उनके कार्यो को पूरा करने के लिए दिए जाते हैं

Database कई प्रकार के access control प्रदान करता है:

- Discretionary Access Control (DAC)
- Mandatory (जरूरी) Access Control (MAC)

Backup और Recovery

सभी Database system में Backup की सुविधा होनी चाहिए, ताकि किसी failure के बाद Data को recover (दोबारा प्राप्त) किया जा सके। हमेशा एक तय समय के बाद database की backup copy बनाना और log file बनाना अच्छा होता है और यह भी सुनिश्चित करना जरूरी है कि सभी copies सुरक्षित जगह (location) पर रखी हो। कभी भी Data के खराब (failure) होने की स्थिति में, जिससे database unstable हो जाता है, backup की copy और log file में रखी हुई जानकारी Database को पुरानी स्थिति में बहाल (restore) करने के काम आती है।

Database security database के सभी पहलुओ और सभी Components की सुरक्षा करती है। जिसमे नीचे दिए गए components शामिल है:

- Database में store किया गया data
- Database server
- Database Management system (DBMS)
- Database की अन्य applications

Database की security सामान्यत: database administrator या किसी अन्य सूचना सुरक्षा संबंधी पेशेवर द्वारा planned, Implement (लागू) और Maintain (बनाए रखना) की जाती है।

कुछ तरीके जिनसे database security (सुरक्षा) implement और maintain की जा सकती है वह है:

- Unathorised (अनअधिकृत) पहुँच और इस्तमाल को प्रबिंधित किया जाए, जिसे मजबूत और multifactor, (जहाँ user सही है या नही ये जानने के लिए एक से ज्यादा Factor हो) access और data management नियंत्रण लागू करके किया जा सकता है।
- Database की Load और data storage की क्षमता की testing पहले ही हो जानी चाहिए ताकि database overload की वजह से crash ना हो जाए, जिसे distributes denial of service (DDOS) भी कहते हैं
- Database के server की और Backup के उपकरणो की भौतिक सुरक्षा जैसे चोरी से सुरक्षा या प्राकृतिक आपदाओ जैसे आग लगना, बाढ़ आदि, भी जरूरी है।
- मौजूदा system की जाँच करते रहना भी जरूरी है कि वह किसी जानी या अनजानी दिक्कत के प्रति संवेदनशील तो नही हो गया है,

8.2 DATABASE SYSTEM की चुनौतियाँ

Database के लिए बढ़ते हुए खतरो को देखते हुए नीचू दिए हुए मुद्दे पर ध्यान देना बहुत जरूरी है:

1. Data Quality (Data की गुणवत्ता)

- Database के लिए कुछ ऐसा technical (तकनीकी) और organisational समाधान होना चाहिए, जो Data की गुणवत्ता को परख सके और प्रमाणित कर सके। यह तकनीक ऐसी भी हो सकती है जैसे आजकल बहुत सारी website पर quality stamps लगी हुई आती है। हमे भी एसी तकनीक चाहिए जो प्रभावशाली तरीके से Data की Quality (गुणवत्ता) को परख सके, जैसे Record linkage
- हमारे पास application level पर लागू होने वाली recovery तकनीक भी होनी चाहिए जो अपने आप Incorrect data (खराब data) को repair कर सके।
- ETL (Extracted Transform and load) tool एक ऐसा tool है जो Data को Load करने के लिए Data warehouses में बड़े पैमाने पर इस्तेमाल होता है, और Data की गुणवत्ता की परेशानीयो से जूझता है।

2. Intellectual Property Right (Property का बौद्धिक अधिकार)

जैसे-जैसे Internet और Internet का इस्तेमाल बढ़ता जा रहा है, Data के कानूनी और सूचना के अधिकार को लेकर organization की चिंता बढ़ गई है। इससे निपटने के लिए water-mark तकनीक का इस्तेमाल भी किया जाता है, जो content को अनअधिकृत इस्तेमाल और Duplication और distribution से बचाती है और content (data) पर स्वामित्व बनाए रखने के लिए एक बढ़िया technique सिद्ध हुई है।

पारंपरिक रूप से यह एक बहुत बड़ी domain की उपलब्धता पर निर्भर करते हैं, जिसके अन्दर objects को उनकी जरूरी और महत्त्वपूर्ण properties को बनाए रखने के साथ-साथ alter भी किया जा सकता है।

तथापि इस तरह की तकनीकों की मजबूती को बढ़ाने के लिए research की आवश्यकता है, और एसी तरीको के बारे में study और investigate करने की जरूरत है जो intellectual property right का violation रोक सकते हैं।

3. Database Survivability (Database का बना रहना)

Database system को हमेंशा अपने functions को संचालित रखना चाहिए, यहाँ तक की कम क्षमताओ के साथ भी, सूचनाओं पर हमले जैसी हानिकारक घटनाओ के बाद भी।

एक DBMS को इसके अलावा हमले को रोकने के प्रयास के साथ उसका पता लगाने में भी सक्षम होना चाहिए। और नीचे दिए हुए कार्य करने चाहिए:

- **Confident:** हमे हमलावर को system तक पहुँचने से रोकने के लिए immediate action लेना चाहिए, और problem को आगे बढ़ने से रोकना चाहिए।
- **Damage assesment:** नुकसान का मूल्यांकन: यह पता लगाना है कि नुकसान किस हद तक हुआ है जिसमे corrupted data और failed function का पता लगाना शमिल है,
- **Recover:** Corrupted और lost data को recover कर सके और खराब हुए function को repair या reinstall कर सके।
- **Reconfiguration:** जब तक recovery की प्रकिया चल रही हो operation (कार्यो) को configure करके degraded mode (एक स्तर नीचे या safe mode) पर चलाना जहाँ सारी functionality नही मिले पर काम चलता रहे।
- **fault treatment:** जहाँ तक संभव हो सके, हमले (attack) में कमजोरी क्या रही उसका पता लगना और आगे के लिए उसे दोबारा होने से रोकना।

8.3 AUTHENTICATION (प्रमाणिकता)

User की प्रमाणिकता यह सुनिश्चित करने के लिए है कि जो व्यक्ति Database का इस्तेमाल कर रहा है वह वही है जो वह बता रहा है। प्रमाणिकता (Authentication) operating system के स्तर पर भी हो सकती है और database के स्तर पर भी हो सकती है। इसके लिए कई system होते हैं जैसे Retina Scanner (आखों की पुतली को scan करना) या biometric (ऊंगली या अंगूठे के निशान) जससे unathorised व्यक्ति database को access ना कर सके।

यहाँ कुछ प्रकार के authentication process दिए गए है:

- Operating system पर आधारित authentication
- Light weight directory access protocol (LDAP)

8.4 AUTHORISATION (प्रधिकार) (अधिकार देना)

Authorisation data base administrator द्वारा दिए गए विशेष अधिकार हैं। Database का कोई user उतना ही हिस्सा या content देख सकता है जितने के लिए व अधिकृत है, बाकी का database उनको देखने की इजाजत नहीं होती।

Authorization के लिए कुछ अलग-अलग तरह की permission (अनुमति) दी जाती है जैसे:

1. **Primary Permission:** यह user की सार्वजनिक रूप से और Directly (सीधे) दी जाती है।
2. **Secondary Permission:** यह किसी विशेष समूह को दी जाती है और अगर user उस समूह का सदस्य है तो उसे यह Permission अपने आप मिल जाती है।
3. **Pubic Permission:** यह सार्वजनिक रूप से सभी user को दे दी जाती है
4. **Content Sensitive Permission:** यह संवेदनशील सूचना से संबंधित होती है और कुछ चुने हुए लोगो को ही मिलती है

Authorisation की चार श्रेणियाँ होती है:

1. **System Administrator:** यह किसी user को दिए जाने वाले सबसे ज्यादा administrative right (प्रबंधन) के अधिकार है। जिसके पास यह अधिकार होता है, वह database को restore और upgrade भी कर सकता है।
2. **System Control:** यह किसी user को दिए जाने वाले उच्चतम (higher) नियंत्रण के अधिकार है। यह database को maintain करने की अनुमति देते हैं परन्तु Data का access नही देते।
3. **System Maintenance:** यह निचले स्तर के नियंत्रण के अधिकार है। यह भी user को database manager के रूप में ही, database की maintenance की अनुमति देते हैं।
4. **System Monitor:** इसको इस्तेमाल करके user database पर निगरानी रख सकता है और उसके snapshots ले सकता है।

8.4 DATABASE INTEGRITY (अखंड़ता)

किसी database में data integrity का मतलब Data का सही होना (correctness), तर्क संगत होना (consistent) और पूरा होना होता है, Data Integrity को नीचे दिए गए तीन constraints लगाकर लागू किया जा सकता है:

1. **Entity Integrity:** यह primary key के concept से संबंधित है। प्रत्येक table की अपनी primary keys होनी चाहिए जो उसकी rows को uniquely Identify कर सके और यह NULL नही होनी चाहिए।
2. **Referential Integrity:** यह foreign keys के concept से संबंधित है, यह किसी relation (table) की वह key होती है जो दूसरे table से referred होती है
3. **Domain Integrity:** इसका मतलब की Database के प्रत्येक column के लिए Domain defined होनी चाहिए।

8.5 SECURITY (सुरक्षा) और INTEGRITY के लिए CONSTRAINTS

Database में store किए गए Data का अनाधिकृत उपयोग (unathorized access), Data का नुकसान, Data में बदलाव आदि से बचाव जरूरी होता है।

Security Violations (सुरक्षा उल्लंघन)

Database का गलत इस्तेमाल दो श्रेणियो में बाँटा जा सकता है: Intentional (जानबूझकर) और Accidental (आकस्मिक)/Accidental loss transaction करते वक्त system के crash होने से हो सकता है।

- Database को बार बार access करने से पैदा हुई विसंगतियां।

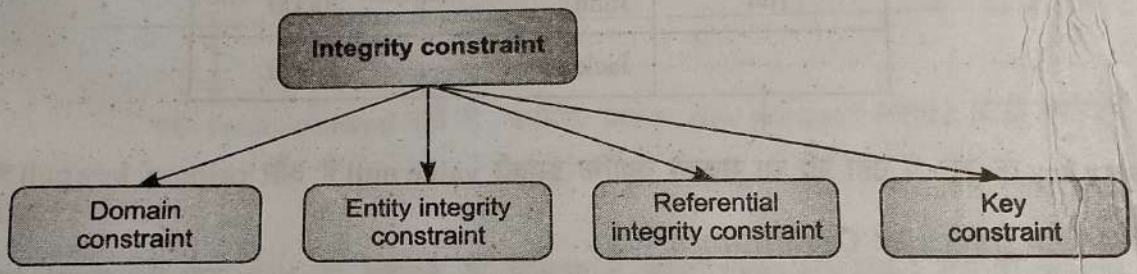
- Data का अनेक Computer पर फैला होने के कारण पैदा हुई विसंगतियां।
- Logical errors से पैदा हुई विसंगतियां/Data consistency के accidental loss को जानबूझकर किए गए नुसान की अपेक्षा रोकना आसान होता है।

Malcious (जानबूझकर या दुर्भावनापूर्ण) access के कई रूप होते हैं जैसे:

- अनाधिकृत रूप से Data को पढना (सूचना की चोरी)
- अनाधिकृत रूप से Data को बदल देना।
- अनाधिकृत रूप से Data को हटा देना (Destroy कर देना)।

Integrity constraints नियमो का समूह होते हैं। यह सूचना की गुणवत्ता को बनाए रखते हैं। यह सुनिश्चित करते हैं कि Data को insert करना, up date करना और बाकी प्रक्रियाएं इस तरह होनी चाहिए कि .Data की अखंडता (Integrity) बनी रहे। इस प्रकार Integrity constraints को database की आकस्मिक नुकसान से सुरक्षा के लिए प्रयोग किया जाता है।

Integrity Constraint के प्रकार



1. Domain Constraints

Domain constraints किसी attribute के लिए valid (वैध) values के set को परिभाषित करते हैं। किसी Domain का Data type: string, character Integer, time, date, currency इत्यादि हो सकता है। Attribute की value हमेश उसकी domain में उपलब्ध होनी चाहिए।

उदाहरण:

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

यहाँ Age के लिए A allowed नही है क्योंकि Age एक संख्या होती है, कोई वर्ण या अक्षर नहीं हो सकती।

2. Entity Integrity Constraints

Entiy Integrity constraints यह बताता है कि primary key की value कभी भी null (खाली) नही हो सकती। क्योंकि primary key की value ही किसी relation में एक विशेष row को identify (पहचानने) के काम आती है, और primary key की value null होगी तो हम उन rows को Identify नही कर पाएगें। एक table में primary key को छोडकर कोई और value null हो सकती है,

उदाहरण:

Employee

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

यह primary key के रूप में नहीं ली जा सकती क्योंकि इसकी value null है और primary key null नहीं हो सकती।

3. Referential Integrity Constraints:

Referential integrity constraint दो tables के बीच specified होता है। सह बताता है कि किसी table 1 में कोई foreign key है जो table no. 2 की किसी primary key से संबंधित है, तो table 1 की प्रत्येक foreign key की value या तो table 2 में उपलब्ध होनी चाहिए या null होनी चाहिए।

उदाहरण:

Table 1

EMP_NAME	NAME	AGE	D_No ← Foreign key
1	Jack	20	11
2	Harry	40	24
3	John	27	18 ← यह value allowed नहीं है, क्योंकि यह table no. 2 में नहीं है।
4	Devil	38	13

Relationships

Table 2

Primary Key → D_No | D_Location

D_No	D_Location
11	Mumbai
24	Delhi
13	Noida

4. Key Constraints:

Keys entity का set होता है जिन्हे किसी entity को set में से पहचानने के लिए इस्तेमाल किया जाता है। एक entity set में अनेक प्रकार की key हो सकती है, परन्तु उनमे से एक key primary key होनी चाहिए। primary key की value unique (अलग) होनी चाहिए और null होनी चाहिए।

उदाहरण

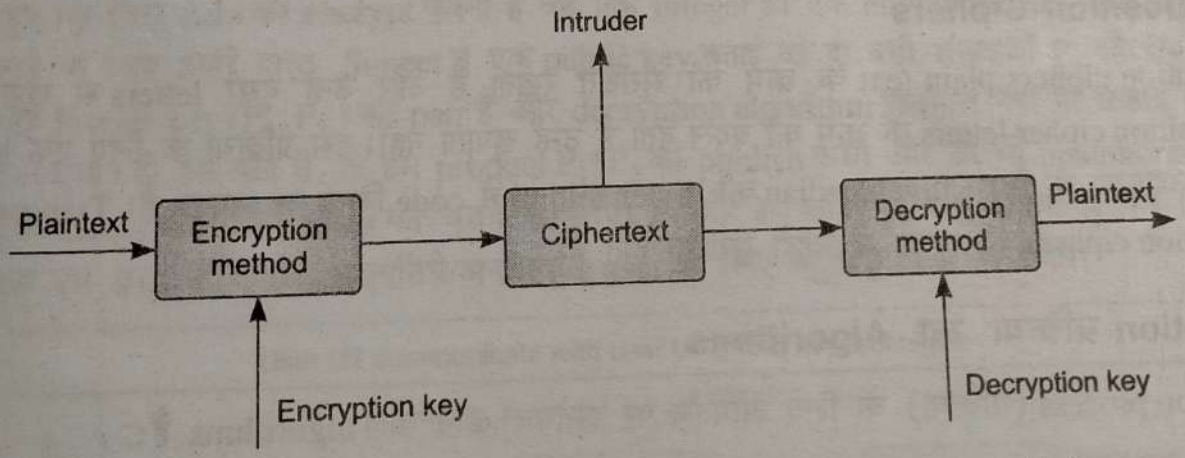
ID	NAME	SEMESTER	AGE
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1002	Morgan	8th	22

यह value allowed नही है, क्योंकि प्रत्येक row unique (अलग) होनी चाहिए।

8.6 ENCRYPTION:

यह Data या सूचना को एक code में बदलने के प्रक्रिया है ताकि Data अनाधिकृत रूप से access ना हो सके। DBMS encryption को information को सुरक्षित रखने के लिए इस्तेमाल कर सकता है, जब DBMS का सामान्य सुरक्ष तंत्र (mechanism) काम नही आए। Data encryption data को meaningless cipher text (किसी ना समझने वाले code) में बदल देता है और बाद में **decryption** से इस code को वापस से original data में बदला जा सकता है। Database encryption को file या column के स्तर पर लागू किया जा सकता है।

इस प्रकार encryption data को private रखने की एक तकनीक है।



Encryption में जिस message को encrypted किया जाता है उसे plain text कहते हैं। Plain text को एक function के द्वारा परिवर्तित किया जाता है, जो एक key के द्वारा parameterized (मापने का तरीका) होता है। Encryption process के output को cipher text कहते हैं। cipher text को तब network में transmit कर दिया जाता है। Plain text को cipher text में बदलने की प्रक्रिया को Encryption कहते हैं, और cipher text को वापस

plain text में बदलने की प्रक्रिया को Descryption कहते हैं। Encryption को transmitting end पर perform किया जाता हैं, और decryption को receiving end पर perform किया जाता है। Encryption की प्रक्रिया के लिए हमें encryption key चाहिए होती है और decryption process के लिए decryption key चाहिए होती है जैसे figure में दिखाया गया है। Decryption की जानकारी के बिना intruder (घुसपेठियां) cipher text को plain text में तोड़ नहीं सकता। इस प्रक्रिया को Cryptography कहते हैं।

Encryption के पीछे basic idea (मूल विचार) encryption algorithm को लागू करना है, जो intruder के द्वारा और original data द्वारा accessible हो और user द्वारा निर्धारित या DBA द्वारा निर्धारित encryption key द्वारा accessible हो जो secret रखी गई हो। Algorithm का output, data का encrypted version होता है। इसके अलावा एक decryption algorithm भी होता है, जो encrypted data और decryption key को input के रूप में लेता है और original data को वापस कर देता है। सही decryption key के बिना, decryption algorithm अस्पष्ट data दे सकता है। Encryption और Decryption keys एक जैसी भी हो सकती है और अलग भी हो सकती है, परन्तु दोनें के बीच कोई संबंध अवश्य होना चाहिए, जो गुप्त होना चाहिए।

Encryption के लिए इस्तेमाल होने वाली तकनीकें

Encryption प्रक्रिया के लिए नीचे दी गई तकनीक इस्तेमाल की जाती है:

- Substitution Ciphers
- Transposition Ciphers

Substitution Ciphers

Substitution cipher में प्रत्येक वर्ण (letter) या letter के समूह को दूसरे letter और letter के समूह से बदल दिया जाता है, छुपाने (गुप्त रखने) के लिए। उदाहरण के लिए: a को D से बदल दिया जाता है, b को E से बदल दिया जाता है, c की F से बदल दिया जाता है, इसी algorithm से प्रत्येक letter को बदला जा सकता है। इसमें attack का मतलब होगा *DWWDFN*. Substitution ciphers बहुत ज्यादा सुरक्षित नहीं होते क्योंकि intruder (घुसपेठ करने वाला) इसे आसानी से समझ सकता है।

Transposition Ciphers

Substitution ciphers plain text के क्रम को सुरक्षित रखता है और उन्हें दूसरे letters से छुपा देता है: परन्तु transposition cipher letters के क्रम को बदल देता है उन्हें छुपाता नहीं। इस प्रक्रिया के लिए एक key का इस्तेमाल होता है। उदाहरण के लिए: iliveinqadian की divienaniqnli में code किया जा सकता है। Transposition ciphers substitution ciphers की तुलना में ज्यादा सुरक्षित होते हैं।

Encryption प्रक्रिया का Algorithms

Encryption process (प्रक्रिया) के लिए आमतौर पर इस्तेमाल होने वाले algorithms हैं:

- Data Encryption Standard (DES)
- Public Key Encryption

Data Encryption Standard (DES)

यह characters का substitution (दूसरे chracter से बदल देना) और rearrangement (क्रम बदल देना) दोनों का इस्तेमाल करता है। जो encryption key पर आधारित होता है। इस तरीके की सबसे बड़ी कमजोरी यह है कि authorised users को encryption key बताना जरूरी है और इस सूचना को communicate करते वक्त चालाक intruders (घुसपेठिया) इसका पता लगा सकते हैं।

Public Key Encryption

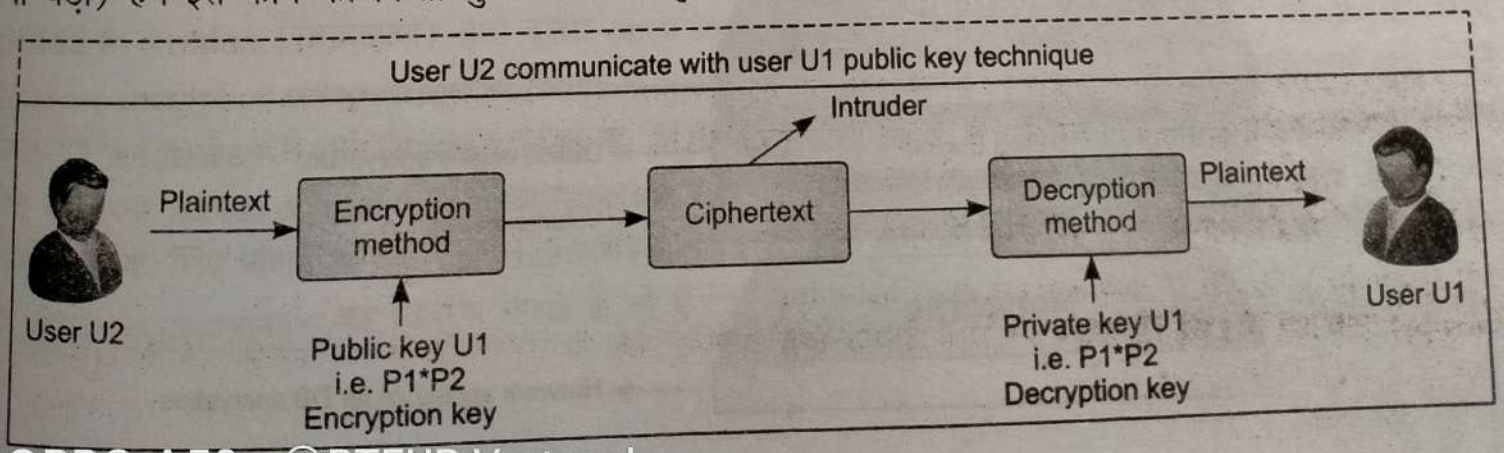
Encryption का दूसरा तरीका है public-key encryption, जो आजकल बहुत प्रसिद्ध हो रहा है। Rivest, Shamir और Adheman द्वारा सुझाया गया encryption scheme जिसे RSA के नाम से जाना जाता है, public-key encryption का एक जाना-माना उदाहरण है। प्रत्येक authorized user के पास एक public encryption key होती है, जो सबको पता होती है, और एक private decryption key होती है (जिसे decryption algorithm द्वारा इस्तेमाल किया जाता है।) जो user के द्वारा चुनी गई होती है और सिर्फ उसे ही पता होती है। यह मान लिया जाता है कि encryption और decryption algorithms publicly known होते है (सबको पता होते हैं)

मान लिजिए कोई user है Suneet और कोई भी Suneet को एक secret message भेज सकता है, Suneet की publicly encryption key का इस्तेमाल करके। अब इस message को केवल Suneet ही decrypt कर सकता है क्योंकि decryption algorithm को Suneet की decryption keys चाहिए होगी जो सिर्फ Suneet को पता है। क्योंकि प्रत्येक user की अपनी चुनी हुई decryption key होती है, इससे DES की कमजोरी को दूर किया जा सकता है।

Public-key relation का एक मुख्य मुद्दा यह है कि encryption और decryption keys कैसे चुनी जाएँ। तकनीकी रूप से public-key encryption algorithms one-way functions के अस्तित्व पर निर्भर करता है, यह वह function होते है जिनको inverse को compute (गणना) करना बहुत मुश्किल होता है।

उदाहरण के लिए RSA algorithm इस observation पर आधारित है कि: यह जानना तो आसान होता है कि दिया गया कोई number (संख्या) prime number (अभाज्य संख्या) है या नहीं परन्तु non-prime number के prime factors निकालना बहुत मुश्किल होता है। एक number जिसमे 100 से ज्यादा अंक हो उसके prime factor निकालने में आजकल के fast computers को भी CPU time के कई साल लग जाऐंगे)

अब हम यह sketch (चित्र) बना सकते है कि RSA diagram के पीछे क्या Intuition (सोच/सहज बोध) है, यह मानते हुए कि जिस data को encrypt करना है वह एक integer हैं। एक encryption key और एक decryption key को चुनने के लिए हमारे दोस्त Suneet ने एक public key बताई जो दो बड़ी संख्याओं P_1 और P_2 का गुणनफल (product) है। Private key (P_1, P_2) का pair है और decryption algorithm इस्तेमाल नहीं हो सकता क्योंकि P_1, P_2 का product (गुणा) ही हमें पता है, तो हम product $P_1 * P_2$ को publish करेंगे और जो भी unauthorised user होगा, उसे $P_1 * P_2$ के factor करने होगें, data को चुराने के लिए। P_1 और P_2 को बहुत बड़ी संख्या चुनकर (100 अंको से भी बड़ी) हम इस काम को किसी घुसपेठिये/अनधिकृत user के लिए बहुत मुश्किल बना देंगे।



हालांकि यह तकनीक बहुत सुरक्षित है, परन्तु गणना करने की दृष्टि से बहुत महंगी है। सुरक्षित communication के लिए एक मिली-जुली scheme इस्तेमाल की जाती है, जिसमें DES Keys को public key encryption के जरिए exchange किया जाता है और बाद में transmitted किए गए data पर DES encryption इस्तेमाल किया जाता है।

Encryption के नुकसान

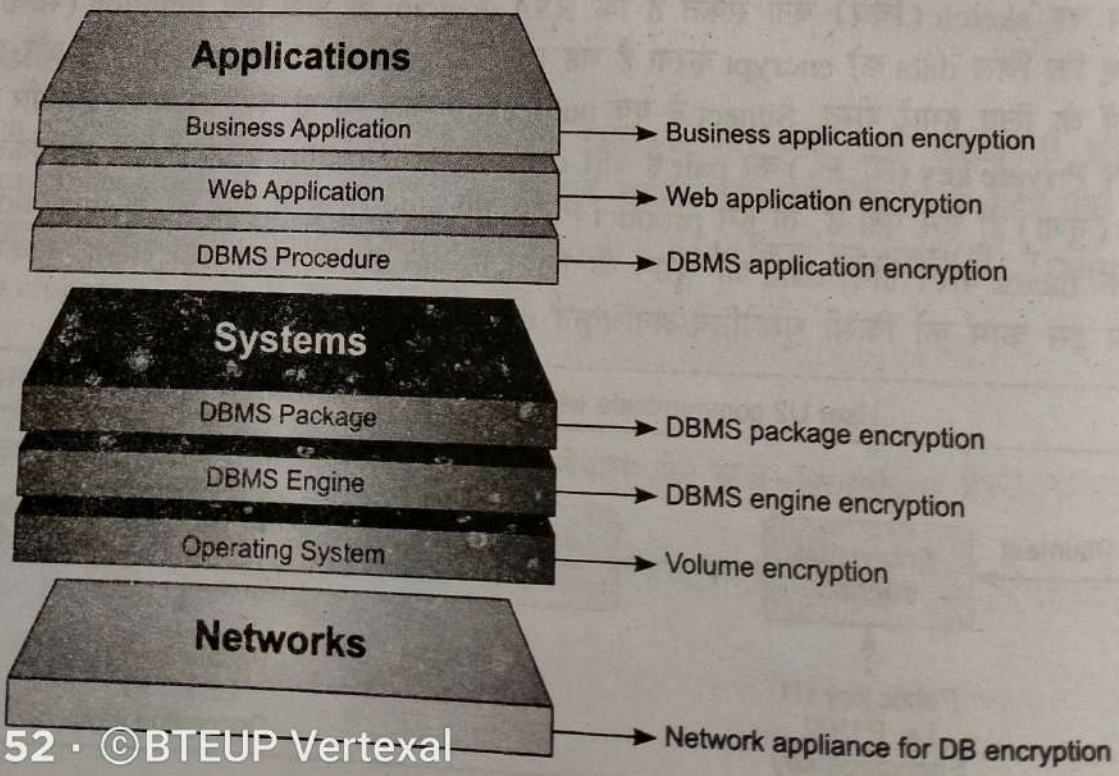
Encryption में निम्नलिखित परेशानियाँ है:

- Key management (मतलब key को secret रखना) एक परेशानी है। यहाँ तक की public-key encryption के तरीके में भी decryption key को सुरक्षित रखना पड़ता है।
- एक system जो encryption को support करता है, उसमें भी data अक्सर plain text के रूप में process हो जाता है। इस प्रकार संवेदनशील data अभी भी transaction programs के द्वारा access हो सकता है।
- Data को encrypt करने से physical storage organisation के स्तर पर गंभीर तकनीकी समस्या हो सकती है। उदाहरण के लिए, जो data encrypted form में store किया गया हो, उस पर indexing करना बहुत मुश्किल होता है।

Database encryption के तरीको के प्रकार

बाजार में बहुत तरह के database encryption methods उपलब्ध है। तब भी बहुत से लोग यह निर्णय नहीं ले पाते कि उनके DBMS environment के लिए कौन-सा solution सबसे बढ़िया होगा। इसका पता हम लगा सकते है अगर हम अलग-अलग तरह के encryption method को समझ ले और यह भी जान ले कि कौन-सा method database के अलग-अलग स्तर पर और अलग-अलग environment में कैसे लागू (apply) होता है।

चार मुख्य तरह के database encryption है: Business Application Encryption (BA), DBMS Application Encryption (DA), DBMS Package Encryption (DP) और DBMS Engine Encryption (DE). प्रत्येक database encryption अलग-अलग स्तर पर काम करता है और अक्सर उनकी functionality (कार्यक्षमता) और requirements (जरूरते) अलग-अलग होती है। नीचे दिए गए चित्र में अलग-अलग स्तर दिखाए गए हैं; जहाँ encryption को रखा जा सकता है (लागू किया जा सकता है)



1. Business Application Encryption (BA)

Business application encryption एक प्रकार की DB encryption है जो मौजूदा term 'API Method' के जैसी है।. BA encryption/decryption API को application या business application server पर लागू करती है, और यह सभी DBMS पर बिना किसी प्रतिबंध के लागू होता है। हालांकि BA encryption method DBMS पर कोई अतिरिक्त भार नहीं डालता, फिर भी encryption की प्रक्रिया बहुत time-consuming (समय लगाती है) होती है क्योंकि encryption data को database में डालने से पहले modify करना (बदलना) पड़ता है।

2. DBMS Application Encryption (DA)

यह database encryption का वह तरीका है, जो encryption/decryption को, एक DBMS product module को API की form में लागू करके, perform करता है। BA application की तरह यह भी अनेक प्रकार के DBMS पर लागू हो सकता है। इसमें query को modify करने का तरीका समान ही होता है, परन्तु encryption/decryption का compute करने का Burden (बोझ) database server पर नहीं होता क्योंकि इसके लिए एक DBMS administration tools होता है। इसके अलावा, DA network environment के किसी network section में सुरक्षा की दृष्टि से होने वाले खतरों के बारे में भी बताता है। इस encryption method की एक कमी यह है कि इसमें कुछ level तक application को भी modify करना पड़ता है।

3. DBMS Package Encryption (DP)

इस method को "Plug-In Method" की तरह इस्तेमाल किया जाता है, जिसमे encryption/decryption को database में product module को install करके किया जाता है। चूंकि module DBMS में एक package के रूप में Installed होता है, DP encryption method encrypted columns की indexing को सहयोग करता है और बिना किसी query modification के आसानी से implement हो जाता है। इसके अलावा GUI (Graphical user Interface) के द्वारा audit function और access control को integrated security function की तरह प्रदान किया गया है। API (Application program interface) से अलग, यह encryption package DBMS में स्वतंत्र रूप से काम करता है और इसमे query के modification की भी कम से कम जरूरत पड़ती है, जो इसे commercial (व्यवसायिक)DBMS और open source DBMS दोनों के लिए उपयोगी बनाता है।

4. DBMS Engine Encryption (DE)

Engine encryption method या DE, DB encryption methods का सबसे विकसित रूप है। क्योंकि encryption/decryption DBMS के engine के level पर perform की जाती है, इसकी implementation सबसे आसान होती है और encryption/decryption की performance सबसे fast होती है। क्योंकि इस तरीके में engine के स्तर (level) पर modification (बदलाव) की आवश्यकता होती है, यह कुछ मामलों में ही इस्तेमाल/प्रदान किया जाता है, पहला DBMS के vendors (विक्रेताओ) द्वारा दूसरा open source DBMS के द्वारा जैसे MySQL और MariaDB तीसरा database encryption companies और database companies के आपसी सहयोग द्वारा। DBMS विक्रेताओं द्वारा प्रदान किए गए solutions जैसे की Oracle's TDE या SQL Server TDE, में केवल encryption के functions उपलब्ध होते है, बिना access control और audit function की उपलब्धता के। Integrated security (स्वीकृत सुरक्षा) को लागू करने के लिए अलग से एक package खरीदना पड़ता है और लागू करना पड़ता है। एक और शब्द जो इस प्रकार के database encryption का विवरण करता है वो है 'Transparent Data Encryption Method'.

Symmetric and Asymmetric Encryption

Symmetric और Asymmetric encryption वह cryptography है जो ciphertext और decryption key के बीच में संबंध को बताता है। यह विचार इस तथ्य (fact) को ध्यान में रखता है कि, users और receiver एकी ही key को share कर सकते है, जो एक सुरक्षित process नहीं है। Encryption algorithms इन दोनों में से (symmetric या asymmetric में से) या तो किसी एक प्रकार का होता है या किसी-किसी मामले में दोनों प्रकार का भी हो सकता है।

- **Symmetric**: इस तरह के case में data को database में save करते वक्त encrypt कर दिया जाता है और जब उसको दोबारा जरूरत होती है तो decrypted कर दिया जाता है। Data को share करने के लिए receiver के पास decryption key की एक copy होनी चाहिए। यह encryption का बहुत सरल और पुराना तरीका है और बहुत जाना-माना भी है। Symmetric encryption की एक कमी यह है कि private key अनुचित तरीके से भी share की जा सकती है और इससे data leak हो सकता है। इस तरह की encryption के उदाहरण है: AES, RC4 और DES.
- **Asymmetric:** यह अपेक्षाकृत कुछ नयी और ज्यादा सुरक्षित प्रकार की encryption है, इसके अंदर private और public key दोनों होती है। Public key किसी के द्वारा भी encryption की अनुमती देती है, परन्तु इस data के बाद में पढ़ने के लिए private key की जरूरत पड़ती है (यह अलग-अलग user के लिए अलग-अलग हो सकती है)। यह communication के दौरान database sharing के लिए ज्यादा सुरक्षित समझा जाता है क्योंकि private keys को share करने की जरूरत नहीं पड़ती। Asymmetric encryption को RSA, DSA, और PKCS के लिए इस्तेमाल किया जाता है।

किसी तरह के Database Encryption ज्यादा Secure होते हैं

जो customers अपने database को encrypt कराना चाहते है, वो इसके सारे विकल्पों (options) के बारे में जानना चाहते हैं। नीचे कुछ database encryption के प्रकार दिए गए है। जिनमें सुरक्षा के अलग-अलग स्तर है और सुरक्षा के साथ-साथ उनकी performance को भी check कर सकते है, ताकि एक ऐसा system चुना जा सके जो practical भी हो और effective भी।

- **AES:** Advanced Encryption Standard एक symmetric algorithm है और बहुत सुरक्षित समझा जाता है। असल में, U.S. government की software से लेकर hardware कंपनियाँ इस algorithm का इस्तेमाल करती है। यह तरीका bit-by-bit stream cipher की जगह एक block cipher का इस्तेमाल करता है, block की lengths 128, 192 या 256 bits की हो सकती है। दूसरे लोगों द्वारा data access करने के लिए users को key share करना जरूरी है, इसका मतलब उन्हें इस को सुरक्षित रखना भी जरूरी है, data का अनाधिकृत प्रयोग रोकने के लिए।
- **RSA:** Rivest-Shamir-Adleman एक asymmetric algorithm है जो encryption के लिए public key का इस्तेमाल करता है और decryption के लिए private key का। यह तरीका आमतौर पर data को unsecured network पर share करते वक्त इस्तेमाल किया जाता है। इसमें key का size 1024 bits से 2048 bits होता है, जो बेहतर सुरक्षा प्रदान करता है परन्तु बाकी methods के मुकाबले इसकी गति काफी धीमी होती है।
- **3DES:** Triple Data Encryption (3DES) एक और block cipher है। यह तीन 56-bit keys का इस्तेमाल करके data को तीन बार encrypt करता है। जिससे एक 168-bit की एक key बन जाती है। यह विकल्प सुरक्षित तो है परन्तु तीन बार encryption होने के कारण इसकी गति धीमी हो जाती है। हालांकि यह बहुत सारे businesses में काम कर रहा है फिर भी एक standard (मानक) की तरह स्वीकार नहीं किया।

- **Two fish:** Two fish भी एक प्रकार का symmetric block cipher है, जिसकी keys की range 128 bits से 256 bits तक है। यह एक flexible तरीका है और विशेषकर यह license-free है। इसकी encryption के rounds की संख्या हमेशा 16 होती है, परन्तु अगर आप key setup या encryption की प्रक्रिया को तेज (Fast) बनाना चाहते है तो वो सुविधा भी इसमें है।

Hashing

अगर सुरक्षा की बात की जाए, तो hashing एक ऐसी तकनीक है, जो data को encrypt करती है और ना समझ पाने वाली hash value को generates करती है। इसमें hash function होता है, जो hash code generate करता है जो data को unauthorised (अनाधिकृत) users के पास जाने से सुरक्षित रखता है।

Hash algorithm यह जाँचने का एक रास्ता भी प्रदान करता है कि जो message मिला है (receive हुआ है) वह वही है कि नहीं जो असल में भेजा गया है। यह plain text message को input के रूप में लेता है और इस पर आधारित एक value दे देता है।

Key Points (मुख्य बिन्दु)

- जो value मिलती है उसकी लंबाई वास्तविक computed message से बहुत कम होती है।
- यह संभव है, कि अलग-अलग text messages एक जैसी value ही generate करे।

यहाँ अब हम hash algorithm का एक sample देखते हैं जिसमें हम message में जितने 'a', 'e' और 'n' के नंबर है उनको गुणा (multiply) कर देगे और उसमें 'o' के नंबर जोड़ देंगे।

उदाहरण के लिए मान लिजिए की message है:

" the combination to the safe is two, seven, thirty-five". अब इस message का hash हमारे simple algorithm को use करके होगा:

$$(2 \times 6 \times 3) + 4 = 40$$

(यहाँ 2 'a' है, 6 'c' है, 3 'n' है और 4 'o' है। ध्यान रहे यह कोई fix algorithm नहीं है यह हमने सिर्फ अपने sample के लिए बताया है)

अब इस message के hash को John के पास cipher text के साथ भेज दिया गया है। जब वह message को decrypted करेगा, वह hash algorithm के हिसाब से इसकी hash value निकाल लेगा। अब अगर यह भेजी गई hash value से नहीं मिलेगी तो वह समझ जाएगा कि message के साथ कुछ छेड़खानी हुई है और उसे बदला गया है।

उदाहरण के लिए John को Bob से एक message मिला उसके साथ hash value 17 थी, जब उसने message को decrypt किया तो वह था "You are being followed, use backroads, hurry"

अब वह अगर इस message की hash value निकालेगा तो वह होगी:

$$(3 \times 4 \times 1) + 4 = 16$$

तो वह समझ जाएगा कि message को बदला गया है, क्योंकि Bob ने जो hash value भेजी थी वह 17 थी।

8.7 DATABASE को RECOVER करने की तकनीक TECHNIQUES

Database systems, किसी अन्य computer system की तरह fail हो सकते हैं, परन्तु उनके अन्दर store किया हुआ data जरूरत पड़ने पर उपलब्ध होना चाहिए। जब भी कोई database fail होता है तो उसमें fast recovery की

सुविधा होनी चाहिए। उसके अंदर atomicity भी होनी चाहिए, मतलब transactions या तो complete होनी चाहिए या तो complete होनी चाहिए या transaction का database पर कोई effect नहीं होना चाहिए, कोई transaction आधी-अधूरी नहीं होनी चाहिए।

Data के backup (data की copy रखना) और recovery (lost data वापस लाना) दोनों के लिए automatic और non-automatic दोनों तरह के तरीके है। जो तकनीक system के crash होने के कारण transaction errors, viruses के कारण catastrophic failure होने से, गलत commands आदि से हुए data के नुकसान को recover करती है वह database recovery techniques होती है। Data के नुकसान को रोकने के लिए कुछ recovery technique जो immediate update, deferred (बाद में) update, और data की backing checkup पर आधारित होती है, इस्तेमाल की जा सकती है।

Recovery techniques एक विशेष file system log पर बहुत ज्यादा निर्भर करती है। यह file किसी transaction की शुरूआत, अन्त या उसमें किसी update की सभी information रखती है। Log उन सभी transaction का, जो database के items की value पर प्रभाव डालती है, पूरा track रखता है। यह सूचना (information) data की transaction failure के बाद recover करने में काम आती है।

- **log को disk में रखा जाता है, start_transaction(T):** यह log entry यह record रखती है कि transaction T ने काम करना शुरू कर दिया है।
- **read_item(T, X):** यह log entry records करती है कि T ने database item X की value read की है।
- **write_item(T, X, old_value, new_value):** यह log entry records करती है कि transaction T ने database item X की value को old_value से new_value में बदल दिया है। Old value को X की before image कहा जाता है और new value को X की after image कहा जाता है।
- **commit(T):** यह log entry record करती है कि transaction (T) पूरी हो गई है और transaction का प्रभाव database में permanently record हो गया है।
- **abort(T):** यह records करती है अगर transaction T को abort (बीच में से वापस) किया गया हो।
- **checkpoint:** checkpoint वह mechanism है जिसमें सभी पुराने logs system से हटाकर storage disk से permanently store कर दिए जाते हैं। checkpoint वह point है जिससे पहले DBMS सही स्थिति में था और सभी transactions पूरी हो गइ थी।

कोई transaction (T) अपने commit point पर पहुच जाती है जब उसके सारे operation जो database को access करते है पूरे हो जाते है, मतलब की वह ऐसे point पर पहुँच जाती है जहाँ से उसे abort करना (बीच में रोकना) मुमकिन नहीं होता। एक बार committed (पूरा) होने के बाद transaction database में permanently record हो जाती है।

Commitment में हमेशा एक commit entry होती है जो log में record होती है और फिर log को disk में रखा जाता है। System के crash होने कि स्थिति में, item को log में search किया जाता है, उन सभी transactions T के लिए जो start entry (T) में recorded थी परन्तु commit entry में recorded नहीं थी। Recovery की प्रक्रिया के दौरान इस तरह की transactions को वापस करके, database पर उनके प्रभाव को खत्म किया जा सकता है।

- **Undoing (वापस करना):** अगर कोई transaction crash हो जाती है, तो recovery manager उसको वापस भी कर सकता है मतलब की उसके operations को reverse भी कर सकता है। इस प्रक्रिया में log entry, write_item(T, x, old_value, new_value) को examine किया जाता है और x की value को old-value

के बराबर set कर दिया जाता है। Non-catastrophic transaction failure से recovery के लिए दो मुख्य तकनीक है: deferred updates और immediate updates.

- **Deferred update:** यह तकनीक तब तक database को disk में physically update नहीं करती जब तक transaction उसके commit point तक नहीं पहुँच जाती। Commit तक पहुचने से पहले transaction के सभी updates local transaction workspace में record होते है। अगर कोई transaction commit point तक पहुचने से पहले ही fail हो जाती है तो, वह database में किसी प्रकार का change (बदलाव) नहीं कर सकती, इसके लिए UNDO की जरूरत भी नहीं पड़ती। जो transactions local transaction workspace में record हुई है उनको REDO करने की जरूरत पड़ सकती है, क्योंकि उनका प्रभाव अभी तक database में record नहीं हुआ। इसलिए deferred update को No-undo/redo algorithm भी कहते है।
- **Immediate update:** इस तरह के update में एक transaction के commit point तक पहुचने से पहले के कुछ operations को database में update कर दिया जाता है। हालांकि यह operation एक log में record हो गए है जो disk पर store है, database पर लागू होने से पहले ही, फिर भी recovery करना संभव है। अगर कोई transaction commit point तक पहुँचने में fail हो जाती है, तो उसके operations के प्रभाव को वापस (undo) करना जरूरी है, मतलब transaction को पूरा वापस करना जरूरी है और हमें undo और redo दोनों करने पड़ेंगें। इसलिए इस तकनीक को undo/redo algorithm कहते है।
- **Caching/Buffering:** इसके अन्दर disk के एक या ज्यादा pages जिनके अन्दर updated होने वाले item है, main memory में cached हो जाते हैं, buffer (बीच में इकठ्ठा) होते है और तब memory में update होते है, disk में दोबारा लिखे जाने से पहले। In-memory-buffers के collection को DBMS की cache कहा जाता है, यह DBMS के control में होती है buffers को hold करने के लिए। यह हिसाब रखने के लिए database के कौन से items buffer में है एक directory का इस्तेमाल किया जाता है। प्रत्येक buffer के साथ एक bit होती है,, वह '0' होगी अगर buffer को modified नहीं किया गया है, और '1' होगी अगर buffer को modified किया गया है।
- **Shadow paging:** यह atomicity (पूरी transaction) और durability (ज्यादा समय तक चलना) प्रदान करती है। एक directory जिसमें 'n' entries होती है create की जाती है, जिसमें i^{th} entry link में database के i^{th} page को point करती है। जब भी कोइ transaction execute होती है current directory को shadow directory में copy कर दिया जाता है। जब किसी page को modified (बदलना) करना होता है, एक shadow page allocated किया जाता है। जिसमें changes किए जाते है और जब वह आगे के उपयोग के लिए तैयार हो जाता है तो सभी pages को original pages में updated कर दिया जाता है।

Some of the backup techniques are as follows:

- **Full database backup:** इसके अन्दर पूरा database, जिसमें data और database system दोनों शामिल होते है, पूरे database को restore करने के लिए Meta information की जरूरत पड़ती है, जिसमें full-text catalogs को backup किया जाता है। एक पहले से defined time series के हिसाब से।
- **Differential backup:** यह सिर्फ data में होने वाले बदलावो को backup (store) करता है, जो आखिरी full database back के बाद हुए है। अगर एक ही data को full backup लेने के बाद बार-बार change किया गया है तो वह उस data के last version को save करता है। इसे करने से पहले हमें एक full database backup करने की जरूरत होती है।

- **Transaction log backup:** इसमें database में होने वाले सभी events (घटनाएँ), जैसे प्रत्येक single transaction जो executed हुई है, को backup किया जाता है। यह transactions की log entries का backup है और इसमें database में होने वाली प्रत्येक transaction शामिल होती है। इससे database एक विशेष समय तक ही recover हो सकता है। इससे transaction log से भी backup ले सकते हैं, अगर files destroyed भी हो गई हो तो और पूरी हुई (committed) एक भी transaction lost नहीं होती।

अभ्यास प्रश्न

1. Database की सुरक्षा (security) क्या होती है?
2. Database security की जरूरत क्यों पड़ती है।
3. Database security के challenges (दिक्कते) क्या है?
4. Data security के लिए क्या-क्या खतरे है।
5. MAC क्या है?
6. Access control को समझाइए?
7. हम database के data को कैसे recover कर सकते है।
8. Hashing क्या होती है?
9. Hashing function के algorithm को समझाइए।
10. AES और RSA को विस्तारपूर्वक समझाइए।
11. Public key encryption का विस्तार से बताइए।
12. Database package encryption (DP) क्या है?
13. किन्हीं दो Backup की तकनीकों को विस्तार में समझाइए।
14. Encryption के नुकसान (disadvantages) बताइए।
15. DES को विस्तार में बताइये।
16. Substitution और transposition ciphers को विस्तारपूर्वक समझाइए।
17. Encryption के अलग-अलग methods को विस्तार से समझाइए।
18. Integrity constraints के बारे में बताइए।
19. Authorization क्या है?
20. Authentication के बारे में बताइए।

❑❑❑

PRACTICALS

1. RDMS में Data definition language (DDL) की command
2. Data manipulation language and (DCL) की command
3. Cursors को use करके PL/SQL programs बनाना
4. PL/SQL में procedures और functions
5. Normalization और E-R Diagram

प्रयोग 1

उद्देश्य–

RDBMS में DDL commands को लागू करना।

COMMANDS:

1. **Command:** CREATE TABLE

Syntax:

```
CREATE TABLE table_name
(
   Column_name [ owner_name ] table_name
);
```

Example Query: SQL> create table student(sno number(8)primary key, sname varchar2(20), fathername varchar2(20), dept varchar2(16), sex char(7), feesdue number(7,2), DOB date);

2. **Command**: ALTER TABLE

Syntax:

```
ALTER TABLE[ owner_name ] table_name
[ADD column_name datatype attributes]
[MODIFY{column_name datatype | column_constraint}]
```

Example Query: SQL>alter table mark modify(total number(5),status char(6));

3. **Command:** DROP TABLE

Syntax:

```
DROP TABLE table name;
```

Example Query: DROP TABLE STUDENT;

Output:

Creating table

Name	*Null?*	*Type*
SNO	NOT NULL	NUMBER(8)
SNAME		VARCHAR2(20)
FATHERNAME		VARCHAR2(20)
DEPT		VARCHAR2(16)
SEX		CHAR(7)
FEESDUE		NUMBER(7,2)
DOB		DATE

Altering Table

Name	*Null?*	*Type*
SNO	NOT NULL	NUMBER(8)
SEM		NUMBER(3)
SUB1		NUMBER(3)
SUB2		NUMBER(3)
SUB3		NUMBER(3)
TOTAL		NUMBER(5)
AVG		NUMBER(5,2)
STATUS		CHAR(6)
GRADE		CHAR(4)

Result:

Thus the DDL commands in RDBMS are implemented.

प्रयोग 2

उद्देश्य–

RDBMS में DML और DCL commands को लागू करना।

COMMANDS:

DML commands:

1. **Command: SELECT**

   Syntax:

```
SELECT[ALL | DISTINCT] select list
FROM table_name1[,…table_nameN]
[JOIN join_condition]
[WHERE search_condition]
```

   Example Query: SELECT * FROM STUDENT WHERE ROLLNO=1;

2. **Command: INSERT**

   Syntax:

```
INSERT INTO[[database_name]owner]{table_name|view_name}
[(column_list)]{[DEFAULT]VALUES|VALUES(value[…])|SELECT
Statement}
```

   Example Query: INSERT INTO STUDENT VALUES(03,"Ramki"," B Tech IT", 87.5, "Distctn" ,"PASS", 12-JUN-1997,"5,Lakshmi nagar,Chennai-24");

   Select * FROM STUDENT;

3. **Command: UPDATE**

   Syntax:

```
UPDATE table_name
SET column_name=expression[,…n]
WHERE search_condition
```

   Example Query: UPDATE STUDENT SET PERCENTAGE=90 WHERE ROLLNO=1;

4. **Command: DELETE**

   Syntax:

```
DELETE[FROM] table_name WHERE search_condition]
```

DCL Commands

1. **Command: GRANT**

   Syntax:

```
GRANT permissions ON objects TO account
```

   Example Query 1: GRANT INSERT ON employee TO PUBLIC;

   Example Query 2: GRANT SELECT,UPDATE ON employee to username;

2. **Command: REVOKE**
Syntax:
```
REVOKE permissions ON object FROM account
```
Example Query: REVOKE SELECT ON student FROM username;

Output:

Selecting from table:

Rollno	*Studentname*	*Dept*	*Average*	*Grade*	*Status*	*Dob*	*Adress*
1	Sudhir	B.tech IT	86	Distctn	Pass	16-JUN-88	82Grace

After Inserting Value

Rollno	*Studentname*	*Dept*	*Average*	*Grade*	*Status*	*Dob*	*Adress*
1	Sudhir	B.tech IT	86	Distctn	Pass	16-JUN-88	82Grace
2	Sudesh	B.tech IT	79.8	Distctn	Pass	26-DEC-87	42 Thinanur
3	Ramki	BTech IT	87.5	Distctn	PASS	12-JUN-1997	5,Lakshmi nagar, Chennai
4	Sudharsan	Mech	77.8	Distctn	Pass	16-JUN-88	3tiruvallur

Result:

Thus the DML and DCL commands in RDBMS are implemented.

प्रयोग 3

उद्देश्य–

PL/SQL programs को पढ़ना और PL/SQL programs को cursor का इस्तेमाल करके लिखना।

PROGRAM:

```
declare
 a number;
 b number;
 su number;
 dif number;
 pro number;
 div number;
begin
a:=&a;
b:=&b;
su:=a+b;
dbms_output.put_line('The Sum is'||su);
dif:=a-b;
dbms_output.put_line('The Difference is'||dif);
pro:=a*b;
dbms_output.put_line('The Product is'||pro);
div:=a/b;
dbms_output.put_line('The Quotent is'||div);
end;
```

Output:

```
Enter value for a: 25
Old 9:a:=&a;
New 9:a:=25;
Enter value for b: 4
Old 10:b:=&b;
New 10:b:=4;
The Sum is 29
The Difference is 21
The Product is 100
The Quotent is 6.25
```

Result:

Thus the PL/SQL program using the cursors were executed successfully.

प्रयोग 4

उद्देश्य–

PL/SQL के procedures और functions को execute करना

PROGRAM:

```
fac number;
i number;
begin
fac:=1;
for I in 1…n
loop
fac:=fac*i;
end loop;
return fac;
end;
//Function created.
declare
n number;
x number;
begin
n:=&n;
x:=factorial(n);
dbms_cuytput.put_line('the factorial of '||n||'is'||x);
end;
```

Output:

SQL>/
Enter value for n: 5
Old 5:n:=&n;
New 5:n:=5;
The factorial of 5 is 120
PL/SQL procedure successfully completed

Result:

Thus the procedures and functions were studied and executed in PL/SQL.

प्रयोग 5

उद्देश्य–

Normalization और E-R Diagram से database design करना

PROCEDURE:

Create a type 'Address' to represent composite attribute.

1. Create the table.
2. Insert the value in the table.
3. Draw the E-R diagram for the table.
4. Convert the given table to the normalized form.
   (a) Converting a table to 1 NF:
   To convert a table to 1 NF remove all the multi valued & composite attributes from the table.
   (b) Converting a table to 2 NF:
      (i) Find and remove attributes that are functionally dependent on only a part of the key and not on the whole key. Place them in a different table.
      (ii) Group the remaining attributes.

NORMALIZING THE TABLE TO 1 NF

```
SQL>create table en1 as select eno, sname, sal from employees;
Table created.

SQL>alter table en1 add primary key(eno);
Table altered.

SQL>create table en2 as select eno,eadd from employees;
Table created.

SQL>alter table en2 add foreign key(eno) references en1(eno);
Table altered.
SQL>desc en2;
```

NORMALIZING THE TABLE TO 2 NF:

```
SQL>create table ep1 as select eno,ename, from empproject;
Table created.

SQL>alter table ep1 add primary key(eno);
Table altered.
```

```
SQL>create table ep2 as select pno,pname from empproject;
Table created.

SQL>alter table ep2 add primary key(pno);
Table altered.

SQL>create table ep3 as select eno,pno,hours from empproject;
Table created.

SQL>alter table ep3 add constraint e3 primary key(eno);
Table altered.

SQL>alter table ep3 add constraint e4 unique(pno);
Table altered.
```

NORMALIZING THE TABLE TO 3 NF:

```
SQL>create table ed1 as select eno,ename,sal,dno from empdept;
Table created.

SQL>alter table ed1 add primary key(eno);
Table altered.

SQL>create table ed2 as select dno,dname from empdept;
Table created.

SQL>alter table ed2 add primary key(dno);
Table altered

SQL>alter table ed1 add foreign key(dno) reference ed2(dno);
Table altered.
```

Result:

Thus the database was designed using E-R diagram and Normailization

MODEL PAPER 1

Maximum marks: 50

Time: 2.30 Hrs

नोट: सभी प्रश्नों के उत्तर दीजिए:

1. नीचे दिए गए प्रश्नों में से किन्हीं दो का उत्तर दीजिए– 2×5

(a) DBMS के 3-level architecture के बारे में बताइए।

(b) Participation constraints को विस्तार से समझाइए।

(c) Relational model को समझाइए।

2. नीचे दिए गए प्रश्नों में से किन्हीं दो का उत्तर दीजिए– 2×5

(a) Mapping constraints के बारे में बताइए।

(b) Projection operation को उदाहरण के मध्य समझाइए।

(c) Normal forms क्या होते हैं?

3. नीचे दिए गए प्रश्नों में से किन्हीं दो का उत्तर दीजिए– 2×5

(a) Functional dependencies को समझाइए।

(b) Join operator को विस्तार से बताइए।

(c) DML, DDL, और DCL को विस्तारपूर्वक समझाइए।

4. नीचे दिए गए प्रश्नों में से किन्हीं दो का उत्तर दीजिए– 2×5

(a) अलग-अलग प्रकार के join को विस्तार से समझाइए।

(b) SQL में views क्या होते हैं?

(c) PL/SQL में user defined functions के बारे में बताइए।

5. नीचे दिए गए प्रश्नों में से किन्हीं दो का उत्तर दीजिए– 2×5

(a) NO-SQL के फायदे और नुकसान बताइए।

(b) Data security में चिताएं क्या हैं?

(c) Authorization और authentication क्या हैं?

❑❑❑

MODEL PAPER 2

Time: 2.30 Hrs *Maximum marks: 50*

नोट: सभी प्रश्नों के उत्तर दीजिए:

1. नीचे दिए गए प्रश्नों में से किन्हीं दो का उत्तर दीजिए– 2 × 5

(a) Database में components क्या-क्या हैं?

(b) विभिन्न प्रकार की keys के बारे में बताइए।

(c) E-R diagram के लाभ और हानि लिखिए।

2. नीचे दिए गए प्रश्नों में से किन्हीं दो का उत्तर दीजिए– 2 × 5

(a) Divide operator को इस्तेमाल करने की condition लिखिए।

(b) ER model को relational model में बदलने के step लिखिए।

(c) BCNF को समझाइए।

3. नीचे दिए गए प्रश्नों में से किन्हीं दो का उत्तर दीजिए– 2 × 5

(a) Relational model constraints को समझाइए।

(b) Decomposition को समझाइए।

(c) Distinct clause को उदाहरण के साथ समझाइए।

4. नीचे दिए गए प्रश्नों में से किन्हीं दो का उत्तर दीजिए– 2 × 5

(a) Constraints क्या हाते है? विस्तार में बताइए।

(b) Procedure क्या होता है?

(c) PL/SQL के loop को उदाहरण सहित बताइए।

5. नीचे दिए गए प्रश्नों में से किन्हीं दो का उत्तर दीजिए– 2 × 5

(a) NO-SQL और RDBMS की तुलना कीजिए।

(b) Database security के challenges (चुनौतियों) को समझाइए।

(c) NO-SQL के फायदे लिखिए।

❑❑❑

MODEL PAPER 3

Time: 2.30 Hrs *Maximum marks: 50*

सभी प्रश्नों के उत्तर दीजिए:

1. नीचे दिए गए प्रश्नों में से किन्हीं दो का उत्तर दीजिए– 2×5

(a) DBMS की विभिन्न languages को समझाइए।

(b) Network model के बारे में विस्तार से बताइए।

(c) E-R diagram की Notations को समझाइए।

2. नीचे दिए गए प्रश्नों में से किन्हीं दो का उत्तर दीजिए– 2×5

(a) Schema और sub schema पर share note लिखिए।

(b) Multivalue dependencies को उदाहरण सहित समझाइए।

(c) Relational algebra में पाँच operations के बारे में बताइए।

3. नीचे दिए गए प्रश्नों में से किन्हीं दो का उत्तर दीजिए– 2×5

(a) Relational model की लाभ और हानि बताइए।

(b) Second normal form (2NF) को समझाइए।

(c) सभी arithmetic operators को उदाहरण सहित समझाइए।

4. नीचे दिए गए प्रश्नों में से किन्हीं दो का उत्तर दीजिए– 2×5

(a) Comparison operators को विस्तार से समझाइए।

(b) PL/SQL के features विशेषताओं को बताइए।

(c) किसी trigger को हम कैसे drop कर सकते हैं?

5. नीचे दिए गए प्रश्नों में से किन्हीं दो का उत्तर दीजिए– 2×5

(a) NO-SQL की पाँच applications को समझाइए।

(b) विभिन्न Integrity constraints को समझाइए।

(c) विभिन्न encryption models को विस्तार से समझाइए।

❑❑❑

EVEN SEMESTER EXAMINATION, OCTOBER-2020

डाटाबेस प्रबन्धन प्रणाली
(Database Management System)

[Time : 2:00 Hours] [Maximum Marks : 50]

नोट–किन्हीं **चार** प्रश्नों के उत्तर दीजिये। सभी प्रश्नों के अंक समान हैं।

1. निम्न में से किन्हीं **दो** के उत्तर दीजिये। [2 × 6¼ = 12½]
   (अ) DBMS क्या है? इसके लाभ लिखिए।
   (ब) निम्न की परिभाषा लिखिए।
   (i) Attributes (ii) Subschema
   (स) आँकड़े के तीन Views की व्याख्या कीजिये।

2. निम्न में से किन्हीं **दो** के उत्तर दीजिये। [2 × 6¼ = 12½]
   (अ) निम्न की व्याख्या कीजिये।
   (i) Super key (ii) Candidate key
   (ब) E-R Model को समझाइये।
   (स) Network model के लाभ व हानि लिखिये।

3. निम्न में से किन्हीं **दो** के उत्तर दीजिये। [2 × 6¼ = 12½]
   (अ) Normalization क्या है? समझाइये।
   (ब) 5NF क्या है? समझाइये।
   (स) निम्न डाटा प्रकार समझाइये—
   (i) Varchar 2 (ii) Raw

4. निम्न में से किन्हीं **दो** के उत्तर दीजिये। [2 × 6¼ = 12½]
   (अ) निम्न को समझाइये।
   (i) DML (ii) DDL
   (ब) निम्न समादेशों की वाक्य-रचना लिखिये।
   (i) Create (ii) Drop
   (स) PL/SQL की विशेषताएँ लिखिये।

5. निम्न में से किन्हीं **दो** पर टिप्पणी लिखिये। [2 × 6¼ = 12½]
   (अ) Triggers
   (ब) Cursors
   (स) Integrity constraints

EVEN SEMESTER EXAMINATION, JULY-2022

डाटाबेस प्रबंधन प्रणाली
(Database Management System)

ne : 2:30 Hours] [Maximum Marks : 50]

–सभी प्रश्न अनिवार्य हैं। प्रत्येक प्रश्नों के कोई **दो** भागों के उत्तर दीजिए।

निम्न में से किन्हीं **दो** के उत्तर दें। [2 × 5 = 10]

(अ) DBMS के Conceptual एवं Internal view को समझाइए।

(ब) DBMS के क्या Components हैं।

(स) DBMS क्या है? लाभ समझाइए।

निम्न में से किन्हीं **दो** के उत्तर दें। [2 × 5 = 10]

(अ) Library Management System का ER diagram बनाइए।

(ब) अलग-अलग Data models के लाभ व हानि लिखिए।

(स) Strong एवं Weak Entity set क्या है? उदाहरण सहित समझाइए।

निम्न में से किन्हीं **दो** के उत्तर दें। [2 × 5 = 10]

(अ) Codd के 12 नियम लिखिए।

(ब) Relation Algebra के सामान्य Operation समझाइए।

(स) Key एवं Integrity Constraints पर संक्षिप्त टिप्पणी लिखिए।

निम्न में से किन्हीं **दो** के उत्तर दें। [2 × 5 = 10]

(अ) Normalization का क्या उद्देश्य है? Updating anomalies को समझाइए।

(ब) Functional dependencies एवं decomposition पर संक्षिप्त टिप्पणी लिखिए।

(स) 1NF, 2NF, 3NF को उदाहरण सहित समझाइए।

निम्न में से किन्हीं **दो** के उत्तर दें। [2 × 5 = 10]

(अ) निम्न SQL Commands को उदाहरण सहित समझाइए: SELECT, WHERE, ORDER BY

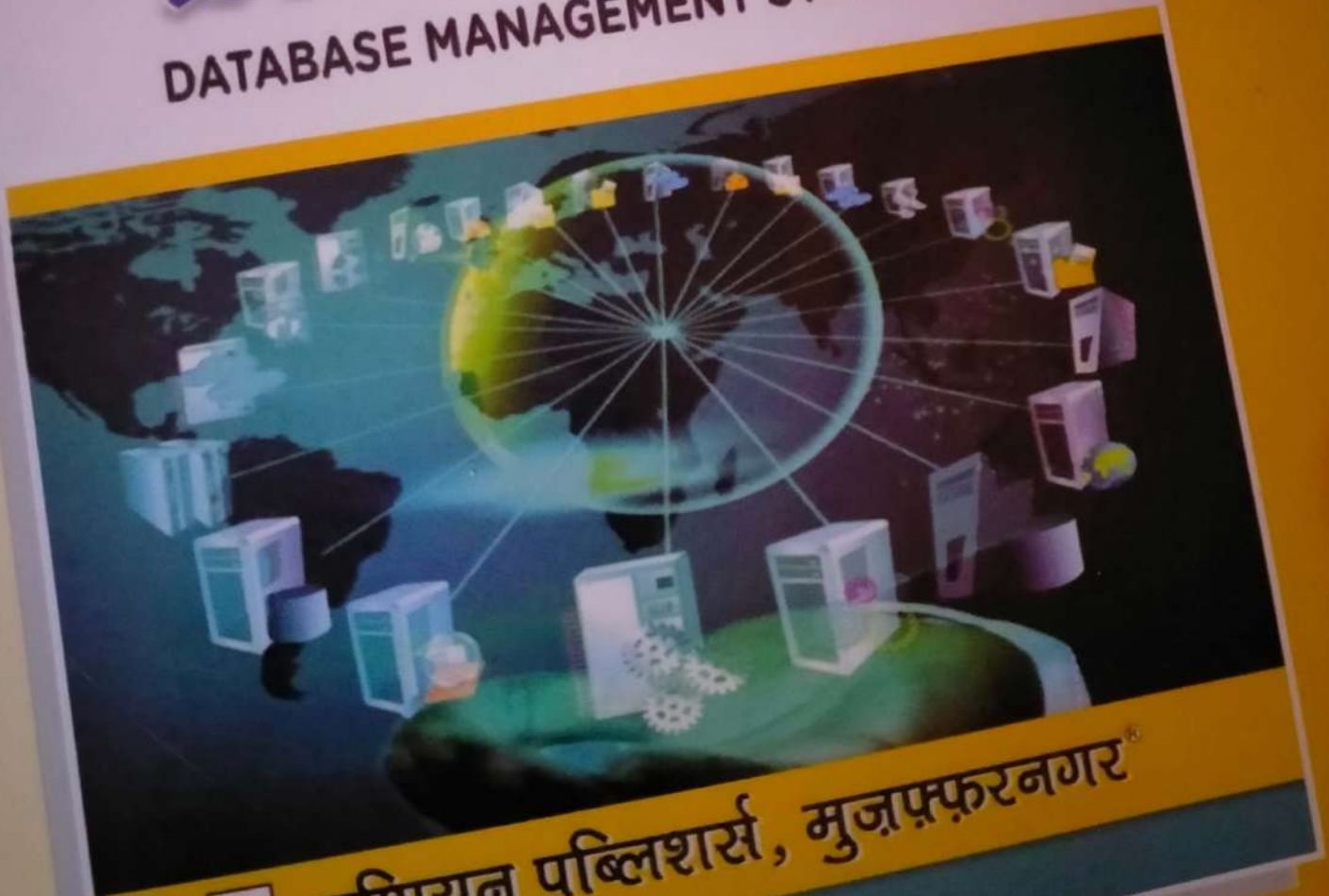
(ब) Security Constraints एवं Integrity constraints पर संक्षिप्त टिप्पणी लिखिए।

(स) PL/SQL में Procedure एवं Stored procedures को समझाइए।

प्रियम तायल • पवन कुमार गुप्ता • प्रद्युम्न कुमार

डाटाबेस प्रबन्धन प्रणाली

DATABASE MANAGEMENT SYSTEM



एशियन पब्लिशर्स, मुज़फ़्फ़रनगर®