DATABASE CONCEPTS

Prof. Deepali Ghorpade

OUTLINE

- Data Objects
- DBMS
  - Applications of DBMS
  - Components of DBMS
  - DBMS Architecture
  - DBMS Design Phases
  - Data Models
- RDBMS
  - Relational keys and Data Integrity
  - Operations on Relations
- SQL

DATA OBJECTS

- Data sets are made up of data objects.
- They are also referred as samples, examples, instances, data points, objects.
- Examples
  - University database: Students, Professors, and Courses.
  - Medical database: Patient, Doctors, Hospital.
  - Sales database: customers, store items, and sales

Patient Table

Patient Id	Name	D.o.B	Gender	Phone	Doctor Id
134	Jeff	4-Jul-1993	Male	7876453	01
178	David	8-Feb-1987	Male	8635467	02
198	Lisa	18-Dec-1979	Female	7498735	01
210	Frank	29-Apr-1983	Male	7943521	01
258	Rachel	8-Feb-1987	Female	8367242	02

Doctor Table

Doctor Id	Doctor	Room
01	Dr Hyde	03
02	Dr Jekyll	06

ATTRIBUTE

- It is a data field, representing a characteristic or feature of a data object.
- It is also referred as dimensions, features, variables.
- *E.g.*

  Customer object attributes: customer_ID, name, and address.
- Types:
  1) Qualitative
     - Nominal
     - Binary
     - Ordinal
  2) Quantitative
     - Numeric
       - Interval-scaled and Ratio-scaled.
       - Discrete and Continuous.

ATTRIBUTE TYPES

Nominal

- categories, states, or "names of things"
- Attribute values do not have any meaningful order among them.
- *E.g*

  hair_color {black, blond, brown, grey, red, white}

  martial_status {single, married, divorced, and widowed}

Binary

- Nominal attribute with only 2 states (0 and 1)
- Symmetric binary: both outcomes equally important
  - E.g. gender: {male, female}
- Asymmetric binary: outcomes not equally important.
  - Convention: assign 1 to most important outcome (HIV positive)
  - E.g. medical_test {positive vs. negative}

ATTRIBUTE TYPES

Ordinal

- Values have a meaningful order (ranking) but magnitude between successive values is not known.
- Discretization of numeric quantities : e.g - *age*
- The central tendency can be represented by mode and median
- *E.g*

  size {small, medium, large}, grades, army rankings

NUMERIC ATTRIBUTE TYPES

- Quantity (integer or real-valued)
- Types
  - **Interval**
    - Measured on a scale of equal-sized units.
    - The differences between values are meaningful.
    - Values have order.
    - No true zero-point
    - E.g. calander dates, temperature in C˚or F˚
  - **Ratio**
    - Inherent zero-point.
    - For ration variables, both differences and ratios are meaningful
    - We can speak of values as being an order of magnitude larger than the unit of measurement (10 K˚ is twice as high as 5 K˚).
    - E.g. temperature in Kelvin, length, counts, age.

NUMERIC ATTRIBUTES TYPES

- **Discrete Attribute**
  - Has only a finite or countably infinite set of values.
  - Sometimes, represented as integer variables (only indivisible values).
  - Note: Binary attributes are a special case of discrete attributes
  - E.g. zip codes, profession, count.
- **Continuous Attribute**
  - Has real numbers as attribute values.
  - It could be divided and reduced to finer and finer levels.
  - Practically, real values can only be measured and represented using a finite number of digits.
  - E.g. distance, age, temperature.

DBMS

- Database
  - Collection of data that is logically coherent.
- Management System
  - Set of programs to store and retrieve those data.
- DBMS (Database Management System)
  - It is a collection of inter-related data and set of programs to store & access those data in an easy and effective manner.
  - It is used to create and maintain the structure of a database, and then to enter, manipulate and retrieve the data it stores.
- Need for Database
  - For otimization of Storage and Retrieval of data.

APPLICATIONS OF DATABASE

- **Telecom**: To keeps track of the information regarding calls made, network usage, customer details etc. Without the database systems it is hard to maintain that huge amount of data that keeps updating every millisecond.
- **Banking System:** For storing customer info, tracking day to day credit and debit transactions, generating bank statements etc.
- **Education sector**: To store and retrieve the data regarding student details, staff details, course details, exam details, payroll data, attendance details, fees details etc.
- **Online shopping**: To store the product information, your addresses and preferences, credit details and provide you the relevant list of products based on your query.
- **Airlines:** reservations, schdules.
- **ERP**(Enterprise resource planning).

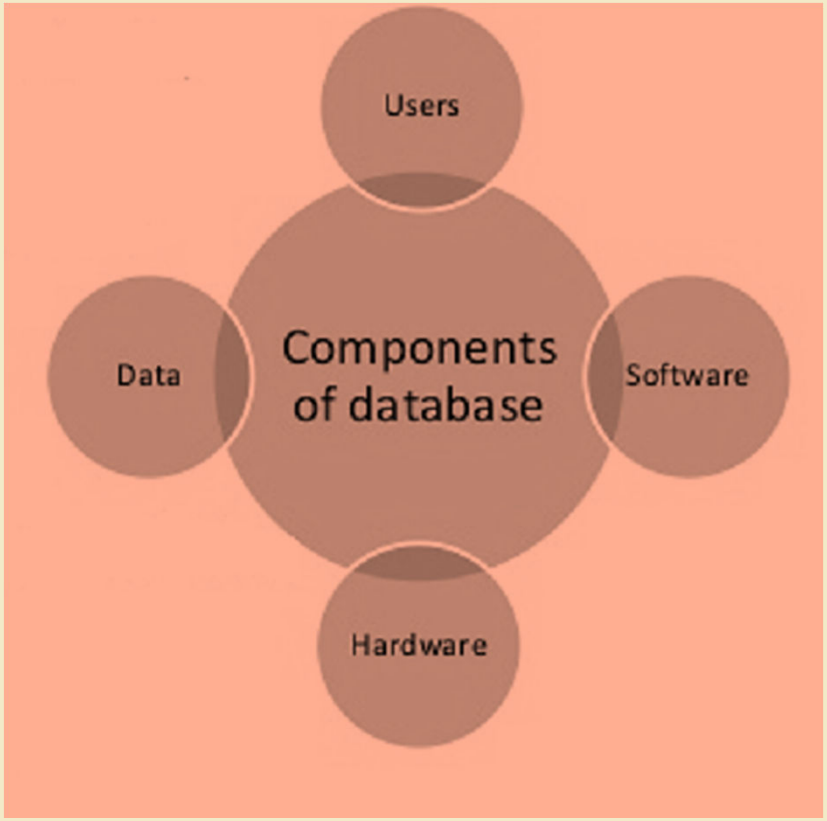
DBMS COMPONENTS

Hardware: Physical device on which database resides. E.g.- computers, disk drives.

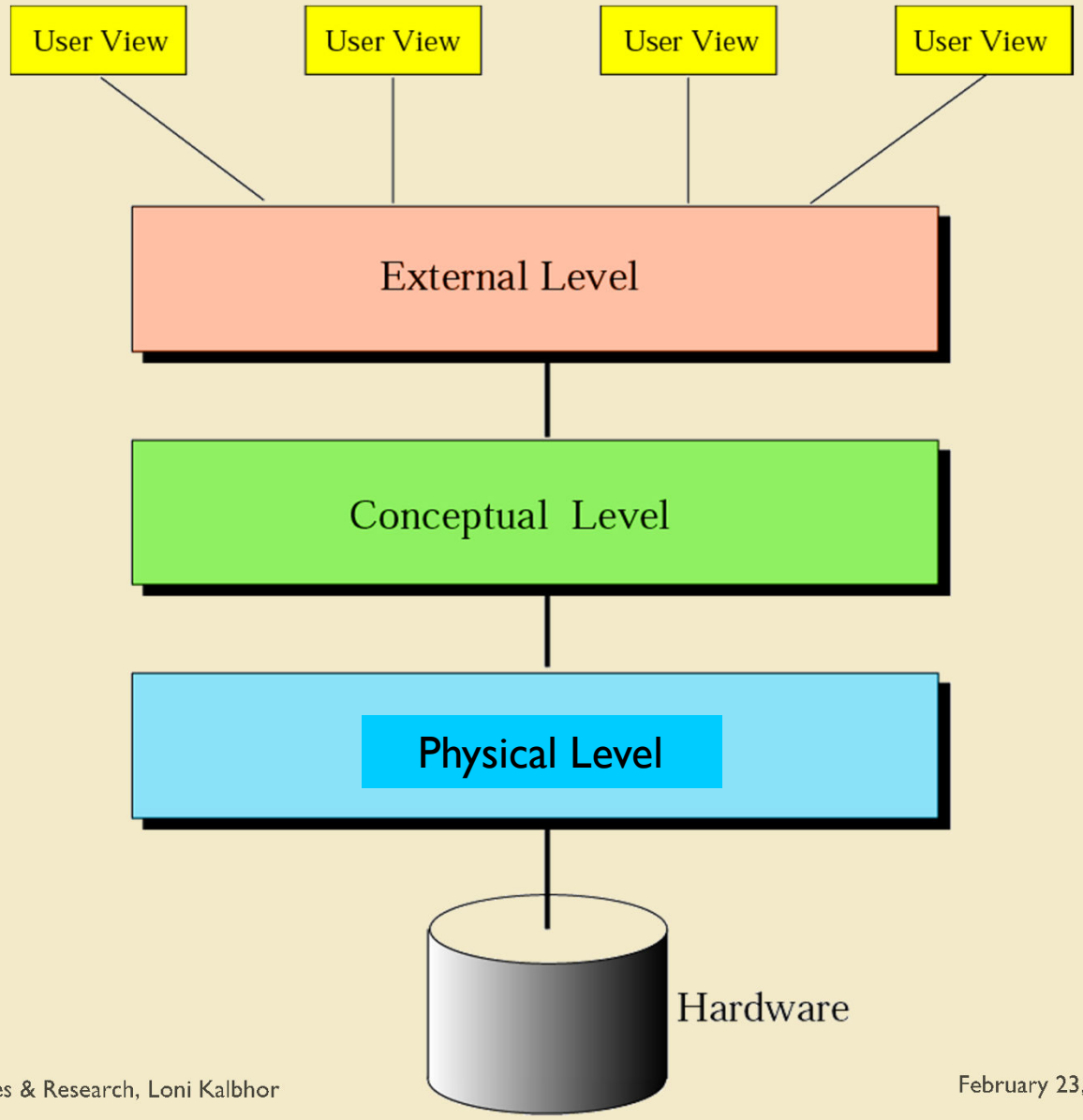
Software: It lies in between user and data. Actual program that allows users to access, maintain, and update physical data.

Data: Stored physically on the storage devices. E.g. numbers, pictures, strings

Users: People who interact with database. E.g. End user, Application programs, Database administrator.



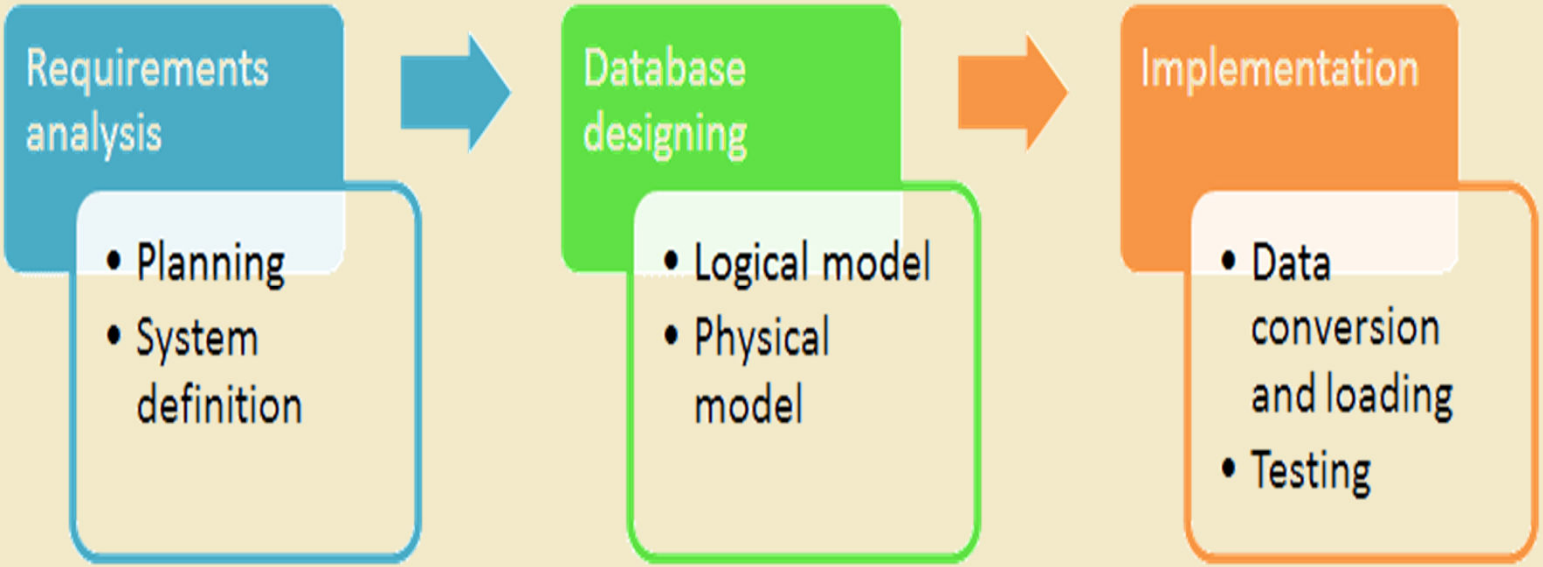
DATABASE ARCHITECTURE



DBMS ARCHITECTURE

- **Physical level**
  - Determines where data is actually stored on the storage device.
  - Low-level access method.
- **Conceptual level**
  - Defines the logical view of the data.
  - The main functions of DBMS are in this level.
- **External level**
  - Interacts directly with the user.
  - Change the data coming from the conceptual level to a format and view that are familiar to the users.

DATABASE DESIGN PHASES



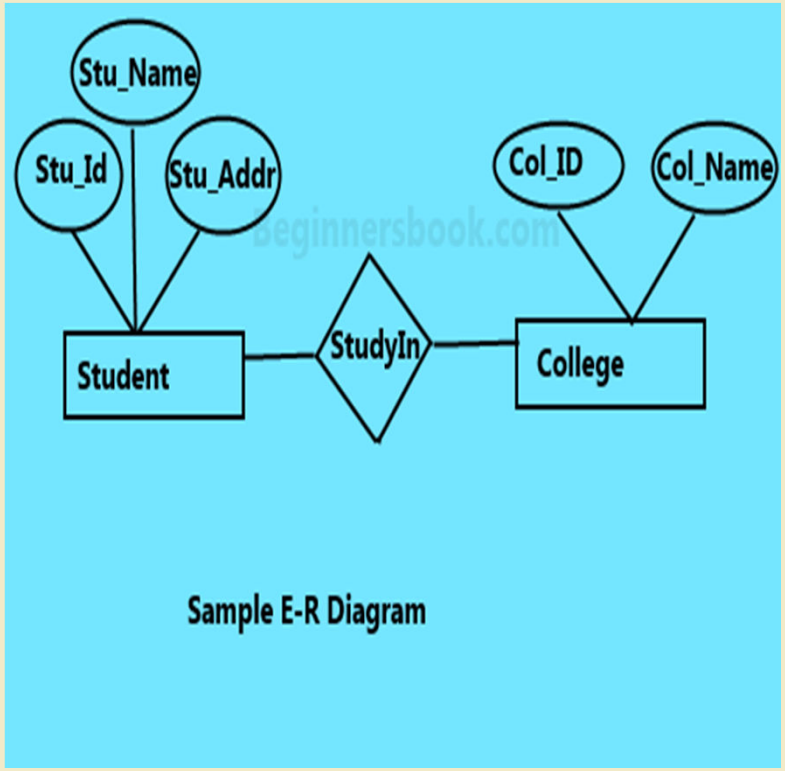
DATA MODELS

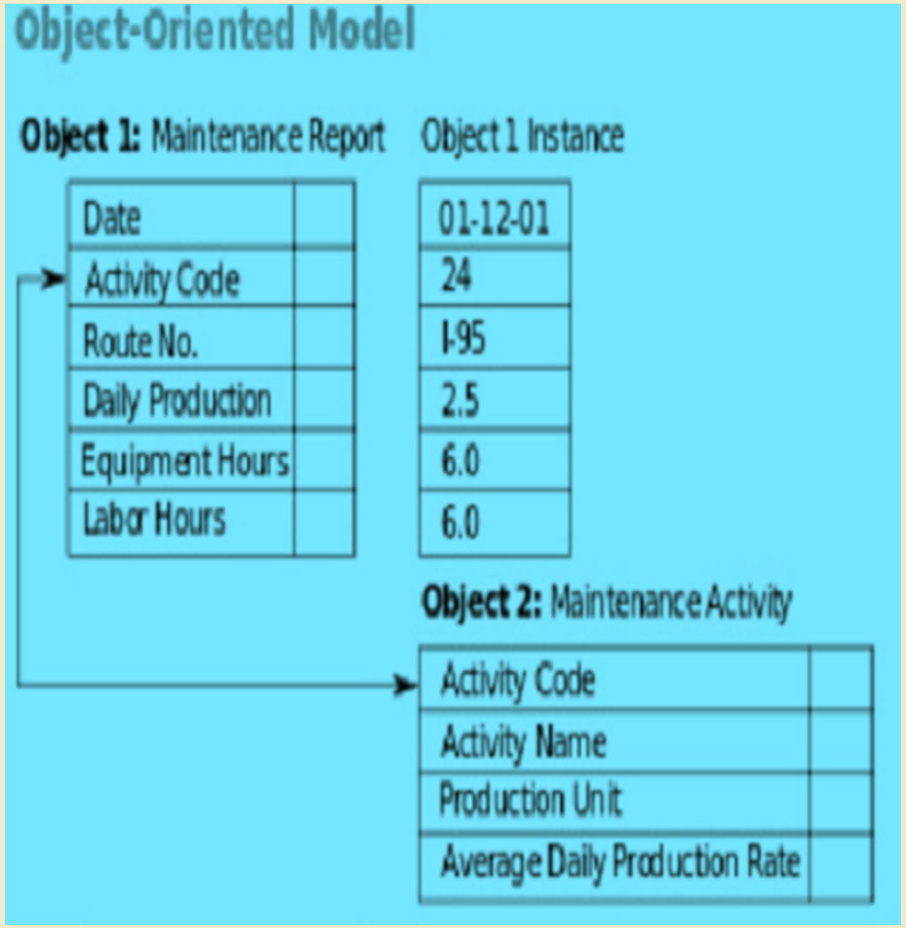
- It is a logical structure of Database.
- It describes the design of database to reflect entities, attributes, relationship among data, constrains etc.
- Basic building blocks

  Entity: Anything about which data is collected or stored.
  Attributes: Characteristics of an entity.
  Relationship: Describes association among entities.
  Constraints: Restrictions placed on data.

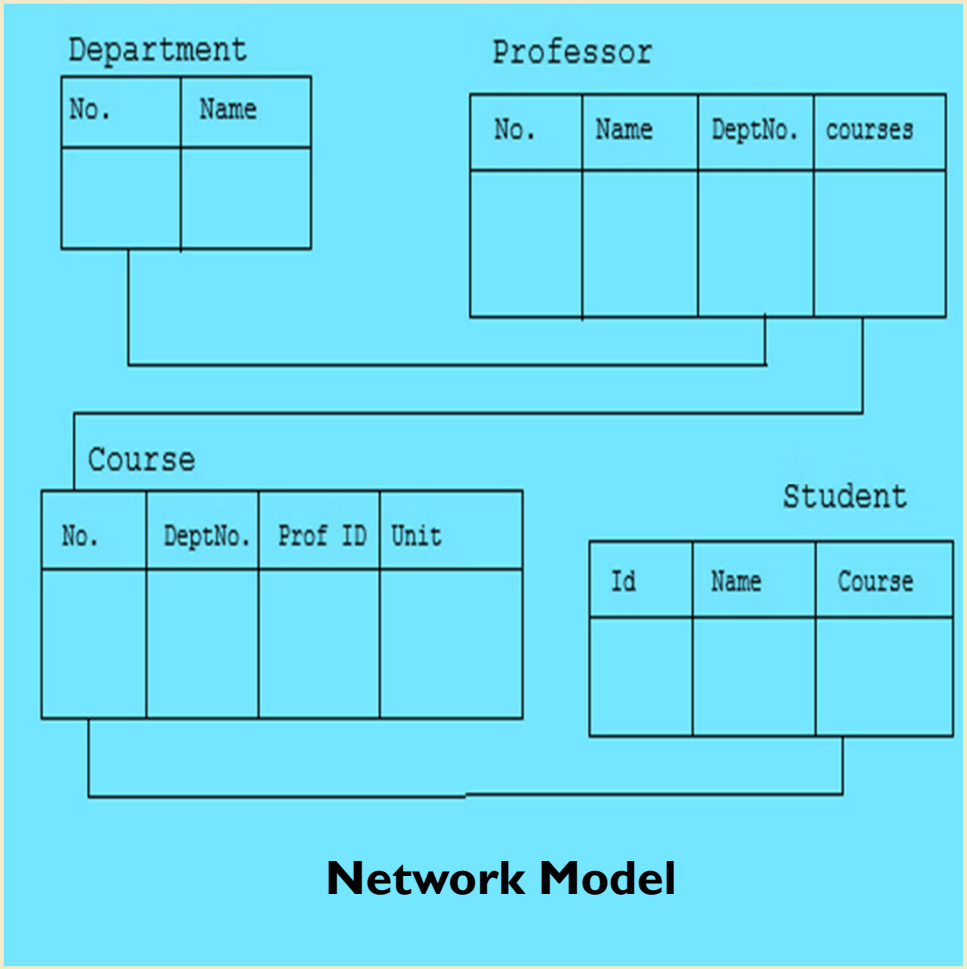
Object based Logical Models	**Record based Logical Models**	**Physical Data Models**
Describe data at the conceptual and view levels.	Describe data at the conceptual and view levels.	These models describe data at the lowest level of abstraction.
1) E-R Model 2) Object Oriented model	1) Relational Model 2) Hierarchical Model 3) Network Model	

DATA MODELS

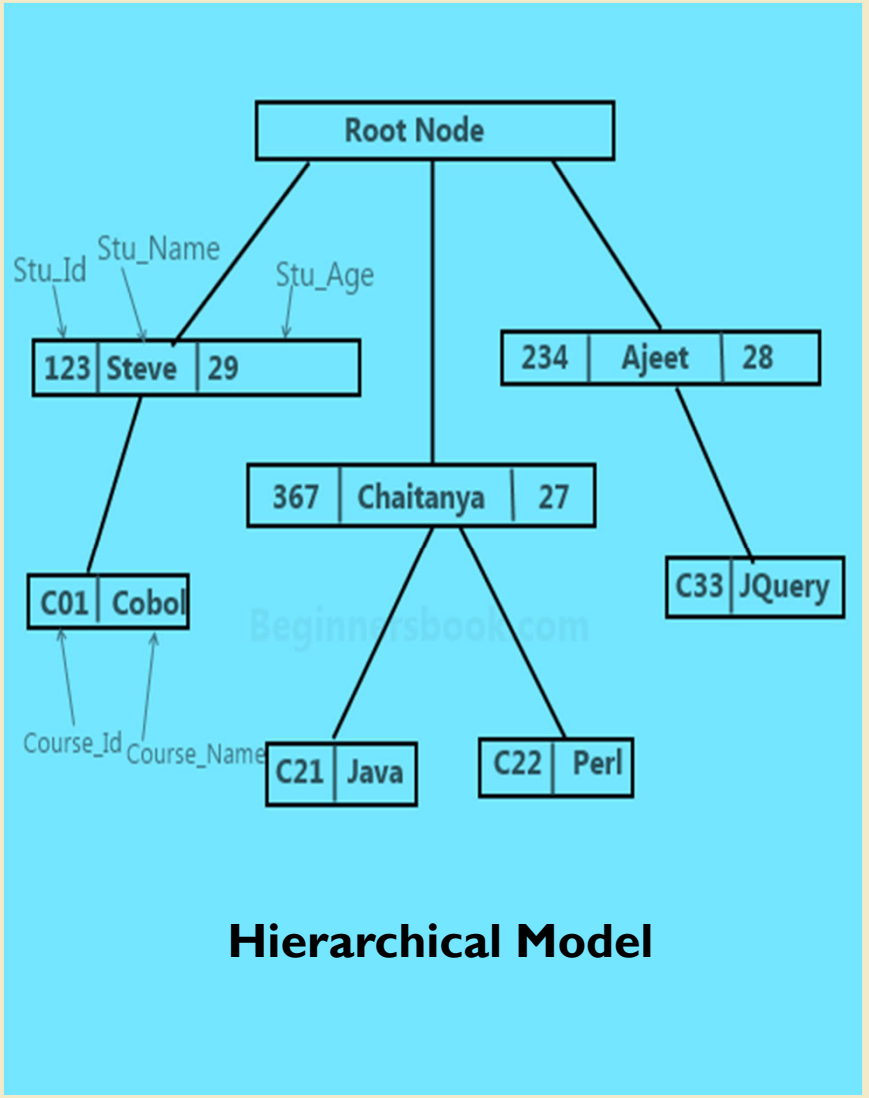

Stu_Name
Stu_Id
Stu_Addr
Col_ID
Col_Name
Student
StudyIn
College
Sample E-R Diagram


Object-Oriented Model
Object 1: Maintenance Report
Object 1 Instance
Date
Activity Code
Route No.
Daily Production
Equipment Hours
Labor Hours
01-12-01
24
I-95
2.5
6.0
6.0
Object 2: Maintenance Activity
Activity Code
Activity Name
Production Unit
Average Daily Production Rate

DATA MODELS



Network Model



Hierarchical Model

DATA MODEL

University Database

Student

stuId	lastName	firstName	major	credits
S1001	Smith	Tom	History	90
S1002	Chin	Ann	Math	36
S1005	Lee	Perry	History	3
S1010	Burns	Edward	Art	63
S1013	McCarthy	Owen	Math	0
S1015	Jones	Mary	Math	42
S1020	Rivera	Jane	CSC	15

The Student Table

Class

classNumber	facId	schedule	room
ART103A	F101	MWF9	H221
CSC201A	F105	TuThF10	M110
CSC203A	F105	MThF12	M110
HST205A	F115	MWF11	H221
MTH101B	F110	MTuTh9	H225
MTH103C	F110	MWF11	H225

The Class Table

Faculty

facId	name	department	rank
F101	Adams	Art	Professor
F105	Tanaka	CSC	Instructor
F110	Byrne	Math	Assistant
F115	Smith	History	Associate
F221	Smith	CSC	Professor

Figure 4.1(b) The Faculty Table

Enroll

stuId	classNumber	grade
S1001	ART103A	A
S1001	HST205A	C
S1002	ART103A	D
S1002	CSC201A	F
S1002	MTH103C	B
S1010	ART103A	
S1010	MTH103C	
S1020	CSC201A	B
S1020	MTH101B	A

The Enroll Table

Relational Model

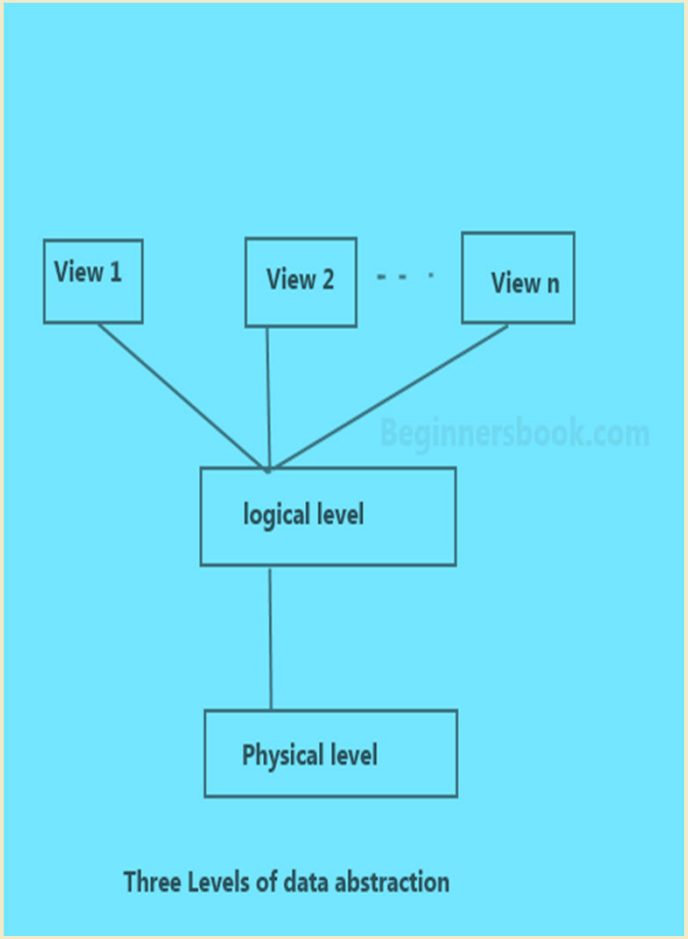
DATA ABSTRACTION

Three levels of abstraction

Physical level: This is the lowest level of data abstraction. It describes how data is actually stored in database. You can get the complex data structure details at this level.

Logical level: This is the middle level of 3-level data abstraction architecture. It describes what data is stored in database.

View level: Highest level of data abstraction. This level describes the user interaction with database system.



Three Levels of data abstraction

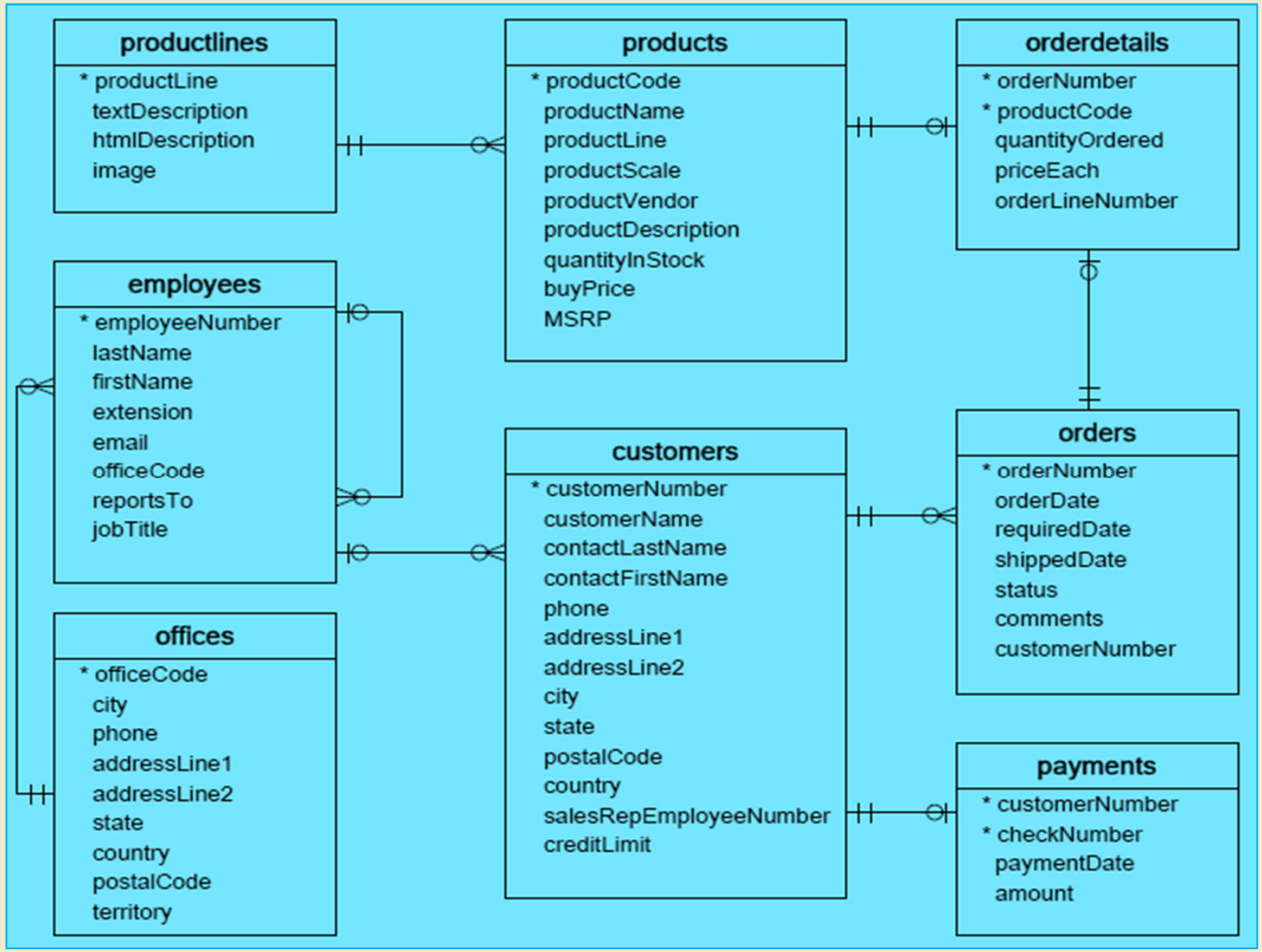
SCHEMA AND INSTANCE

Logical design of a database is called the schema.

- **physical schema:** The design of a database at physical level. It defines how the data stored in blocks of storage at this level.
- **Logical schema:** Design of database at logical level. The programmers and database administrators work at this, at this level data can be described as certain types of data records gets stored in data structures.
- **View schema:** Design of database at view.This generally describes end user interaction with database systems.

Instance: The data stored in database at a particular moment of time is called instance. It is snapshot of data in database at given instance of time

SCHEMA EXAMPLE



RELATIONAL MODEL

- Relational Model was proposed by E.F. Codd to model the data and relationships in form of inter-related tables and represents how data is stored in Relational Databases.
- It is implemented using any RDMBS languages like Oracle SQL, MySQL etc.
- A relational database stores data in the form of relations which is two-dimensional table.

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
2	RAMESH	GURGAON	9652431543	18
3	SUJIT	ROHTAK	9156253131	20
4	SURESH	DELHI		18

Table - STUDENT

RDBMS

- All relational databases can be used to manage transaction-oriented applications (OLTP).
- OLTP databases can be thought of as "operational" databases, characterized by frequent, short transactions that include updates, touch a small amount of data, and provide concurrency to thousands (if not more) of transactions (some examples include banking applications and online reservations).
- RDBMSs support ACID transactions (Atomicity, Consistency, Isolation, and Durability) to maintain data integrity.
- RDBMS is for enterprise OLTP and ACID compliance, or databases under 1 terabyte.
- The most popular RDBMS

  Microsoft SQL Server, Oracle Database, MySQL, and IBM DB2.

TERMINOLOGIES IN RELATIONAL MODEL

- **Domain:** A domain is the set of allowed values for one or more attributes. Each attribute in the model should be assigned domain information that includes: data types, length, date format, range, constraints, null support, default value.
- **Attribute:** Attributes are the properties that define a relation. e.g ROLL_NO, NAME
- **Relation Schema:** A relation schema represents name of the relation with its attributes. e.g. STUDENT (ROLL_NO, NAME, ADDRESS, PHONE and AGE
- **Tuple:** Each row in the relation is known as tuple. The above relation contains 4 tuples.
- **Relation Instance:** The set of tuples of a relation at a particular instance of time is called as relation instance. Table STUDENT shows the relation instance of STUDENT at a particular time. It can change whenever there is insertion, deletion or updation in the database.

TERMINOLOGIES IN RELATIONAL MODEL

- **Degree:** The number of attributes in the relation is known as degree of the relation. The STUDENT relation defined above has degree 5.
- **Cardinality:** The number of tuples in a relation is known as cardinality. The STUDENT relation defined above has cardinality 4.
- **Column:** Column represents the set of values for a particular attribute. The column ROLL_NO is extracted from relation STUDENT.
- **NULL Values:** The value which is not known or unavailable is called NULL value. It is represented by blank space..

Formal Terms	Alternative 1	Alternative 2
Relation	Table	File
Tuple	Row	Record
Attribute	Column	Field

RELATIONAL KEYS

- They identify one or more attributes (called relational keys) that uniquely identify each tuple in a relation to avoid any duplicate tuple within a relation.
- They also establishes relationship among tables.
- Different types of keys
  - Candidate
  - Primary
  - Composite
  - Foreign
  - Super
  - Alternate

RELATIONAL KEYS

Super Key

- It is a attribute or set of attributes or combination of columns in a table that can uniquely identify any database record without referring to any other data.
- {Emp_Id} , {Emp_No}, {Emp_Id, Emp_name},
- {Emp_SSN, Emp_No, Emp_Name},{Emp_No, Emp_Name}

Emp_No	Emp_Name	Emp_SSN
101	Steve	SSN23
102	John	SSN24
103	Robert	SSN28
104	Carl	SSN22

Table - Employee

RELATIONAL KEYS

Candidate Key

- It is minimum number of columns required to uniquely identify each row, or it is minimal super key.
- Each table may have one or more candidate keys.
- {Emp_Id} , {Emp_Number}

Emp_No	Emp_Name	Emp_SSN
101	Steve	SSN23
102	John	SSN24
103	Robert	SSN28
104	Carl	SSN22

Table - Employee

RELATIONAL KEYS

Primary key

- It is one of best among the candidate keys to use, for identification.
- {Emp_Id} or {Emp_Number}

Alternate key

- Out of all candidate keys, only one gets selected as primary key, remaining keys are known as alternative or secondary keys**.**
- If Emp_Id is selected as primary key, then Emp_Number will be super key.

Emp_No	Emp_Name	Emp_SSN
101	Steve	SSN23
102	John	SSN24
103	Robert	SSN28
104	Carl	SSN22

Table - Employee

RELATIONAL KEYS

Composite Key or Compound key

- A key that consists of more than one attribute to uniquely identify rows.
- It is a key composed of more than one column.
- {cust_id, order_id}

cust_Id	order_Id	product_code	product_count
C01	O001	P007	23
C02	O123	P007	19
C02	O123	P230	82
C01	O001	P890	42

Table - Sales

RELATIONAL KEYS

Foreign Key

- Foreign keys are the attributes in a table that points to the primary key of another table. They act as a cross-reference between tables.
- The Stu_Id column in Course_enrollment table is a foreign key as it points to the primary key of the Student table.

Stu_Id	Stu_Name	Stu_Age
101	Chaitanya	22
102	Arya	26
103	Bran	25
104	Jon	21

Table : Student

Course_Id	Stu_Id
C01	101
C02	102
C03	101
C05	102
C06	103
C07	102

Table : Course_enrollment

DATA INTEGRITY

Data integrity

- It is the overall completeness, accuracy and consistency of data.
- This can be indicated by the absence of alteration between two instances or between two updates of a data record, meaning data is intact and unchanged.
- It is usually imposed during the database design phase through the use of standard procedures and rules.
- Data integrity can be maintained through the use of various error-checking methods and validation procedures.

STUDENT_ID	STUDENT_NAME	ADDRESS	SUBJECT
100	Joseph	Alaiedon Township	Mathematics
101	Allen	Fraser Township	Chemistry
100	Joseph	Alaiedon Township	Physics
102	Chris	Clinton Township	Mathematics
103	Patty	Troy	Physics

TYPES OF DATA INTEGRITY

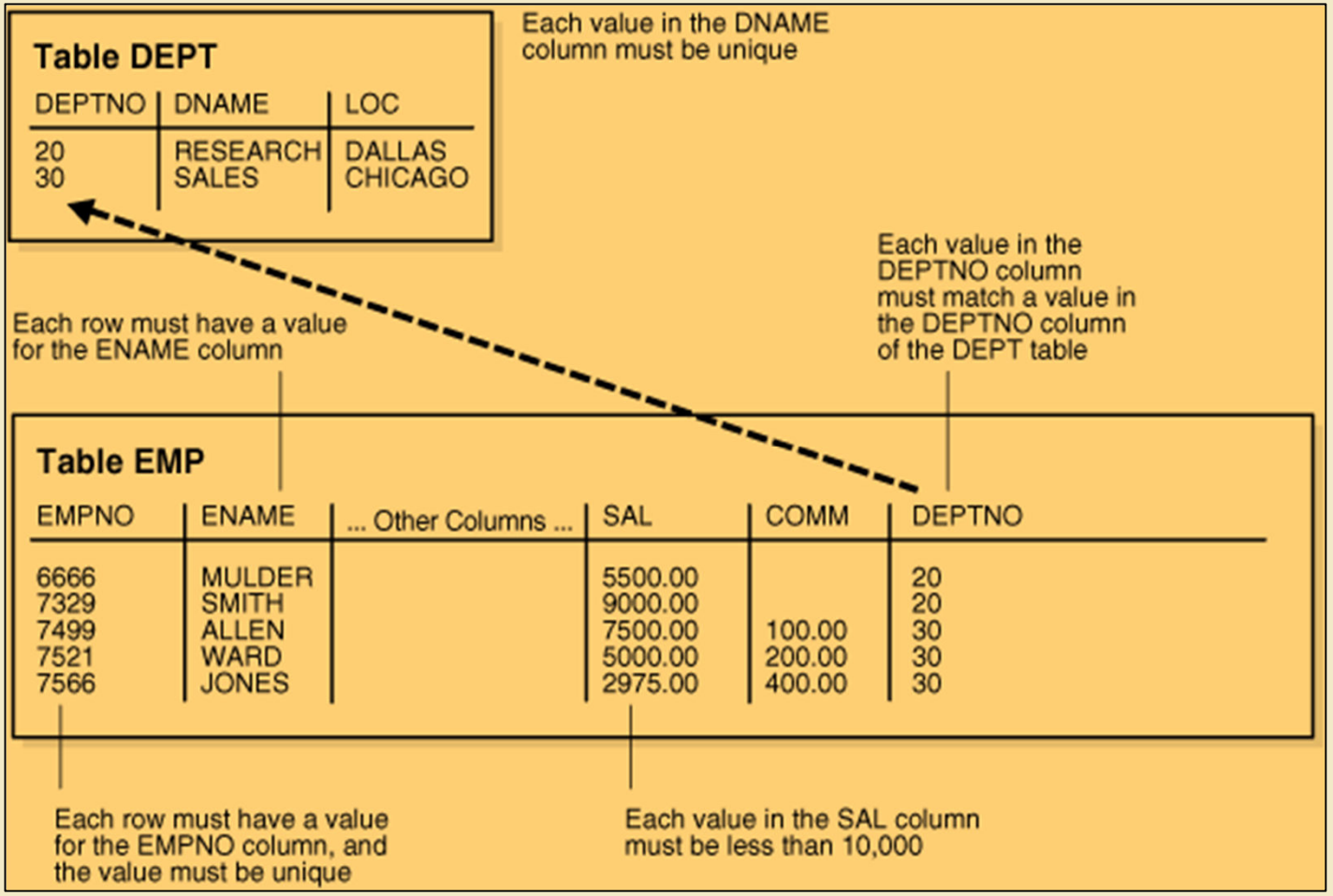
Domain Constraint: This states that all columns in a relational database are in a defined domain.

Entity Constraint: This is concerned with primary keys .The rule states that every relation must have its own primary key and each has to be unique and not null.

Referential Integrity: This is concept foreign keys . A referential integrity rule is a rule defined on a key (a column or set of columns) in one table that guarantees that the values in that key match the values in a key in a related table (the referenced value).The rule states that foreign key can be in two states , either it should refer to primary key of another table or it can be null.

- Restrict: Disallows the update or deletion of referenced data.
- Cascade: When referenced data is updated, all associated dependent data is correspondingly updated. When a referenced row is deleted, all associated dependent rows are deleted.

DATA INTEGRITY



OPERATIONS ON RELATIONS

- In a relational database, we can define several operations to create new relations out of the existing ones.
- It takes instances of relations as input and yields instances of relations as output. They accept relations as their input and yield relations as their output.
- It uses operators to perform queries.
- An operator can be either unary or binary.
- Basic Operations
  - Insert
  - Update
  - Join
  - Union
  - Difference
  - Delete
  - Select
  - Project
  - Intersection

UNARY OPERATION

COURSES

No	Course-Name	Unit
CIS15	Intro to C	5
CIS17	Intro to Java	5
CIS19	UNIX	4
CIS51	Networking	5

→ **Insert** →

No	Course-Name	Unit
CIS15	Intro to C	5
CIS17	Intro to Java	5
CIS19	UNIX	4
CIS51	Networking	5
CIS52	***TCP/IP Protocols***	***6***

COURSES

No	Course-Name	Unit
CIS15	Intro to C	5
CIS17	Intro to Java	5
CIS19	UNIX	4
CIS51	Networking	5
CIS52	TCP/IP Protocols	6

→ **Delete** →

No	Course-Name	Unit
CIS15	Intro to C	5
CIS17	Intro to Java	5
CIS51	Networking	5
CIS52	TCP/IP Protocols	6

COURSES

No	Course-Name	Unit
CIS15	Intro to C	5
CIS17	Intro to Java	5
CIS19	UNIX	4
CIS51	Networking	5
CIS52	TCP/IP Protocols	6

→ **Update** →

No	Course-Name	Unit
CIS15	Intro to C	5
CIS17	Intro to Java	5
CIS19	UNIX	4
CIS51	Networking	**6**
CIS52	TCP/IP Protocols	6

UNARY OPERATION

COURSES

No	Course-Name	Unit
CIS15	Intro to C	5
CIS17	Intro to Java	5
CIS19	UNIX	4
CIS51	Networking	5
CIS52	TCP/IP Protocols	6

No	Course-Name	Unit
CIS15	Intro to C	5
CIS17	Intro to Java	5
CIS51	Networking	5

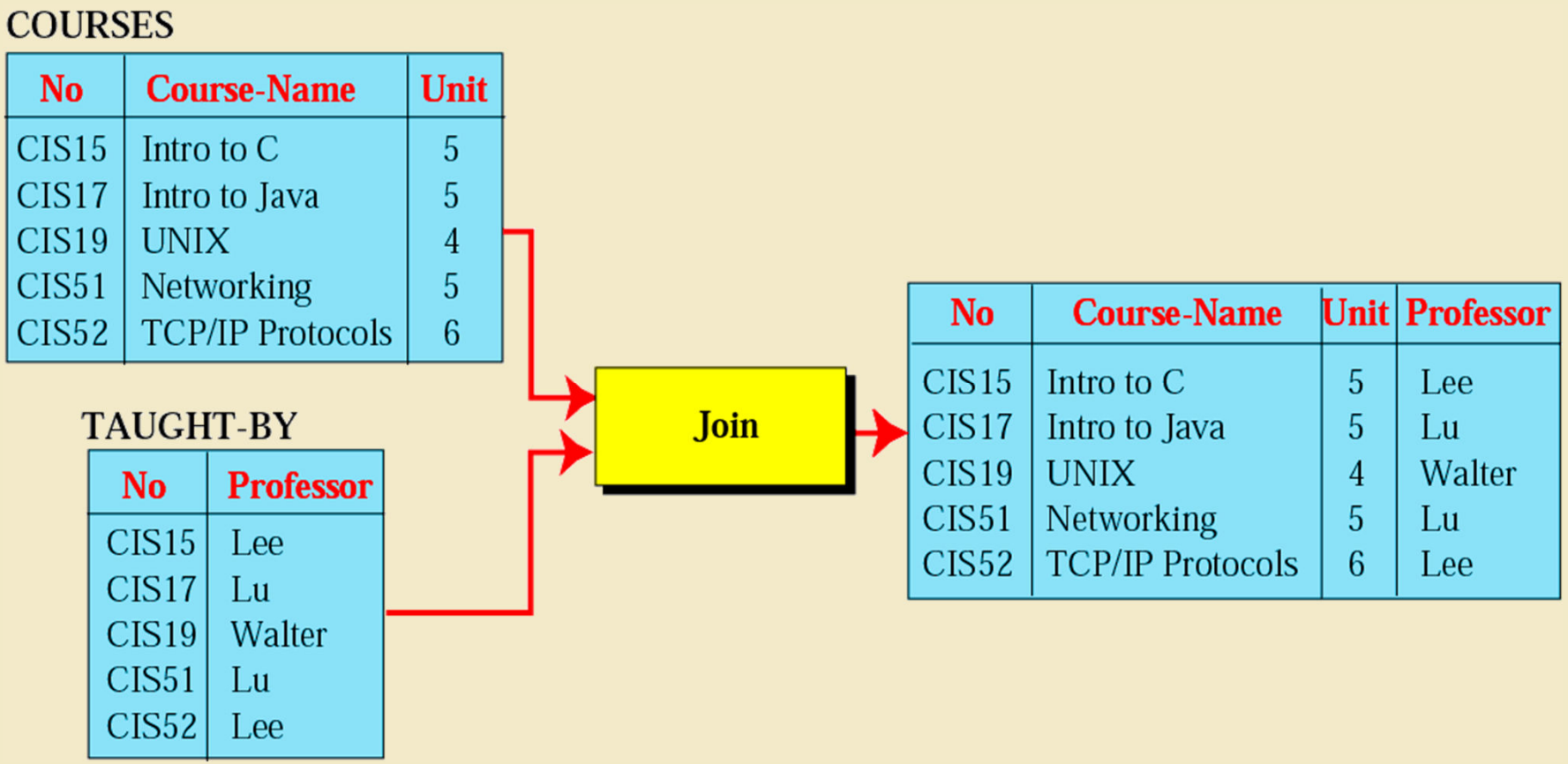
COURSES

No	Course-Name	Unit
CIS15	Intro to C	5
CIS17	Intro to Java	5
CIS19	UNIX	4
CIS51	Networking	5
CIS52	TCP/IP Protocols	6

No	Unit
CIS15	5
CIS17	5
CIS19	4
CIS51	5
CIS52	6

BINARY OPERATION

- Combines two relations based on common attributes.



BINARY OPERATION

- Creates a new relation in which each tuple is either in the first relation, in the second, or in both.
- The two relations must have the same attributes.

CIS15-Roster

Student-ID	F-Name	L-Name
145-67-6754	John	Brown
232-56-5690	George	Yellow
345-89-6580	Anne	Green
459-98-6789	Ted	Purple

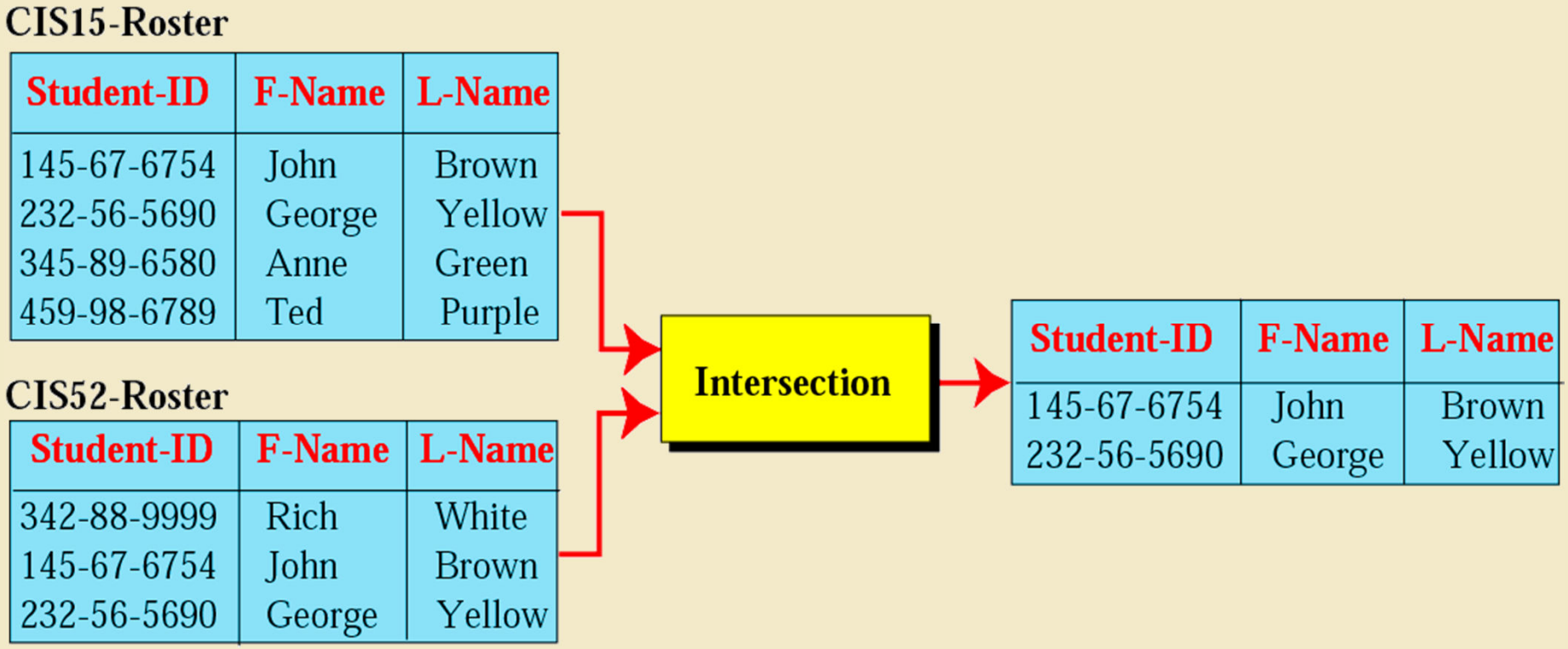
CIS52-Roster

Student-ID	F-Name	L-Name
342-88-9999	Rich	White
145-67-6754	John	Brown
232-56-5690	George	Yellow

Student-ID	F-Name	L-Name
145-67-6754	John	Brown
232-56-5690	George	Yellow
345-89-6580	Anne	Green
459-98-6789	Ted	Purple
342-88-9999	Rich	White

BINARY OPERATION

- Creates a new relation in which each tuple is a member in both relations.
- The two relations must have the same attributes.



DIFFERENCE OPERATION

- Creates a new relation in which each tuple is in the first relation but not the second.
- The two relations must have the same attributes.

CIS15-Roster

Student-ID	F-Name	L-Name
145-67-6754	John	Brown
232-56-5690	George	Yellow
345-89-6580	Anne	Green
459-98-6789	Ted	Purple

CIS52-Roster

Student-ID	F-Name	L-Name
342-88-9999	Rich	White
145-67-6754	John	Brown
232-56-5690	George	Yellow

Student-ID	F-Name	L-Name
345-89-6580	Anne	Green
459-98-6789	Ted	Purple

DIFFERENCE BETWEEN DBMS AND RDBMS

No.	DBMS	RDBMS
1)	DBMS applications store data as file.	RDBMS applications store data in a tabular form.
2)	In DBMS, data is generally stored in either a hierarchical form or a navigational form.	In RDBMS, the tables have an identifier called primary key and the data values are stored in the form of tables.
3)	Normalization is not present in DBMS.	Normalization is present in RDBMS.
4)	DBMS does not apply any security with regards to data manipulation.	RDBMS defines the integrity constraint for ACID (Atomicity, Consistency, Isolation and Durability) property.
5)	DBMS uses file system to store data, so there will be no relation between the tables.	in RDBMS, data values are stored in the form of tables, so a relationship between these data values will be stored in the form of a table as well.

DIFFERENCE BETWEEN DBMS AND RDBMS

No.	DBMS	RDBMS
6)	DBMS has to provide some uniform methods to access the stored information.	RDBMS system supports a tabular structure of the data and a relationship between them to access the stored information.
7)	DBMS does not support distributed database.	RDBMS supports distributed database.
8)	DBMS is meant to be for small organization and deal with small data. it supports single user.	RDBMS is designed to handle large amount of data. it supports multiple users.
9)	Examples of DBMS are file systems, xml etc.	Example of RDBMS are mysql, postgre, sql server, oracle etc.

STRUCTURED QUERY LANGUAGE (SQL)

STRUCTURED QUERY LANGUAGE

- It is a database computer language designed for the storing, retrieving , manipulating and management of data in a relational database.
- SQL is the standard language for Relational Database System.
- It is standardized by ANSI and ISO for use on relational databases.
- It is a declarative language, which involve giving broad instructions about what task is to be completed, rather than the specifics on how to complete it. It deals with the results rather than the process, thus focusing less on the finer details of accomplishing each task.
- SQL commands are instructions, coded into SQL statements, which are used to communicate with the database to perform specific tasks, work, functions and queries with data.
- SQL commands are grouped into three major categories depending on their functionality:
  - Data Definition Language (DDL)
  - Data Manipulation Language (DML)
  - Data Manipulation Language (DML)

SQL COMMANDS

Data Definition Language (DDL):

- Used to define the database structure or schema.
- These SQL commands are used for creating, modifying, and dropping the structure or schema of database objects.
- Commands
  - Create - To create objects in the database
  - Drop - Delete objects from the database
  - Alter - Modifies an existing database object, such as a table.
  - TRUNCATE - To delete tables in a database instance.
  - RENAME - To rename database instances.

Data Control Language (DCL):

- These SQL commands are used for providing security to database objects.
- Commands
  - Grant- Gives user's access privileges to database
  - Revoke- Withdraw access privileges given with the GRANT command

SQL COMMANDS

Data Manipulation Language (DML):

- Used for managing data within schema objects.
- These SQL commands are used for storing, retrieving, modifying, and deleting data in database.
- Commands
  - Select- retrieve data from the database
  - Insert- Insert data into a table
  - Update- Updates existing data within a table
  - Delete- Delete existing rows

DATA DEFINITION LANGUAGE

Syntax	Example
CREATE DATABASE database_name;	**CREATE DATABASE** university;
USE database_name;	**USE student;**
CREATE TABLE table_name(column1 datatype, column2 datatype, ... columnN datatype, PRIMARY KEY(one or more columns));	**CREATE TABLE** StudInfo(Id INT NOT NULL, Name VARCHAR (20) NOT NULL, Age INT NOT NULL, Address CHAR (25) , **PRIMARY KEY** (Id));

DATA DEFINITION LANGUAGE

Syntax	Example
DESC table_name	**DESC** StudInfo;

Syntax	Example
DROP TABLE table_name	**DROP TABLE** StudInfo

Syntax	Example
SHOW DATABASES;	Displays all databases
SHOW TABLES;	Lists all tables from used database

DDL – ALTER CLAUSE

- The ALTER TABLE statement is used to modify the definition (structure) of a table by modifying the definition of its columns.
- The ALTER TABLE statement is also used to add and drop various constraints on an existing table.

Syntax	Example
ALTER TABLE table_name **ADD** column_name datatype;	**ALTER TABLE** CUSTOMERS **ADD** SEX char(1); **ALTER TABLE** CUSTOMERS **DROP** SEX;
ALTER TABLE table_name **MODIFY** column_name column_type;	**ALTER TABLE** customers **MODIFY** ADDRESS char(75);
	ALTER TABLE orders **CHANGE** ID cust_id char(50);
ALTER TABLE table_name **CHANGE COLUMN** old_name **TO** new_name;	**ALTER TABLE** CUSTOMERS **CHANGE COLUMN** Name **TO** Full_name VARCHAR(100);

DDL – TRUNCATE CLAUSE

- It removes all tuples from a table, but the table schema is retained.
- It is used to delete complete data from an existing table.
- The operation cannot be rolled back.
- RENAME command is used to change the name of the table or a database object.

Syntax	Example
TRUNCATE TABLE Table_name	**TRUNCATE TABLE** CUSTOMERS;
RENAME old_table_name **To** new_table_name;	**RENAME** Employee **TO** My_Emloyee;

DML – INSERT CLAUSE

Syntax	Example
INSERT INTO TABLE_NAME (column1, column2,...columnN) **VALUES** (value1, value2,value3,...valueN);	**INSERT INTO** StudInfo (Id, Name, Age, Address) **VALUES** (1,'Siddhant',15, 'Pune');
INSERT INTO TABLE_NAME **VALUES** (value1,value2,value3,...valueN);	**INSERT INTO** StudInfo **VALUES** (2, 'Shubhan', 12, 'Satara');
INSERT INTO table_name [(column1, column2, ... columnN)] **SELECT** column1, column2,...columnN **FROM** table_name [**WHERE** condition];	**INSERT INTO** Employee (id, name, dept, age, salary location) **SELECT** emp_id, emp_name, dept, age, salary, location **FROM** temp_employee;
INSERT INTO table_name [(column1, column2, ... columnN)] **SELECT** * **FROM** table_name.	**INSERT INTO** Employee (id, name, dept, age, salary location) **SELECT** * **FROM** temp_employee;

DML - SELECT CLAUSE

Syntax	Example
SELECT column1,column2,.. columnN **FROM** table_name;	**SELECT** Id, Name FROM StudInfo;
SELECT * **FROM** table_name;	**SELECT** * **FROM** StudInfo;
SELECT DISTINCT column1, column2, ... **FROM** table_name;	**SELECT DISTINCT** * **FROM** StudInfo;

DATA MANIPULATION LANGUAGE

WHERE Clause:

- Condition can be specified using the comparison or logical operators like >, <, =, LIKE, NOT, etc.

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

DML - WHERE CLAUSE

Syntax	Example
SELECT column1, column2, columnN **FROM** table_name **WHERE** [condition]	**SELECT** ID, NAME, SALARY **FROM** CUSTOMERS **WHERE** SALARY **>** 2000;
	SELECT ID, NAME, SALARY **FROM** CUSTOMERS **WHERE** NAME = 'Hardik';
SELECT column1, column2, columnN **FROM** table_name **WHERE** [condition1] **AND** [condition2]...**AND** [conditionN];	**SELECT** ID, NAME, SALARY **FROM** CUSTOMERS **WHERE** SALARY **>** 2000 **AND** age < 25;
SELECT column1, column2, columnN **FROM** table_name **WHERE** [condition1] **OR** [condition2]...**OR** [conditionN]	**SELECT** ID, NAME, SALARY **FROM** CUSTOMERS **WHERE** SALARY **>** 2000 **OR** age < 25;

DML – LIKE CLAUSE

LIKE clause is used to compare a value to similar values using wildcard operators. There are two wildcards used in conjunction with the LIKE operator.

- The percent sign (%) {zero, one or multiple characters}
- The underscore (_) {single number or character}

Syntax	Example
SELECT **FROM** table_name **WHERE** column **LIKE** 'XXX%'	**SELECT** ID, NAME, SALARY **FROM** CUSTOMERS **WHERE** SALARY **LIKE '200%'**
SELECT **FROM** table_name **WHERE** column **LIKE** '%XXX%'	**SELECT** ID, NAME, SALARY **FROM** CUSTOMERS **WHERE** SALARY **LIKE '%500%'**

DML – LIKE CLAUSE

Syntax	Example
SELECT **FROM** table_name **WHERE** column **LIKE** '_XX%'	**SELECT** ID, NAME, SALARY **FROM** CUSTOMERS **WHERE** SALARY **LIKE '_00%'**
SELECT **FROM** table_name **WHERE** column **LIKE** '2X%'	**SELECT** ID, NAME, SALARY **FROM** CUSTOMERS **WHERE** SALARY **LIKE '2_%'**
	SELECT ID, NAME, SALARY **FROM** CUSTOMERS **WHERE** SALARY **LIKE '2___3'**
	SELECT ID, NAME, SALARY **FROM** CUSTOMERS **WHERE** SALARY **LIKE '_2%3'**

DML – BETWEEN OPERATOR

- The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.
- The BETWEEN operator is inclusive: begin and end values are included.

Syntax	Example
SELECT *column_name(s)* **FROM** *table_name* **WHERE** *column_name* **BETWEEN** *value1* **AND** *value2;*	**SELECT** * **FROM** CUSTOMERS **WHERE** AGE **BETWEEN** 22 AND 27;
	SELECT * **FROM** CUSTOMERS **WHERE** AGE **NOT** **BETWEEN** 22 AND 27;

DML – DELETE AND UPDATE

Syntax	Example
DELETE **FROM** table_name **WHERE** condition;	**DELETE** **FROM** CUSTOMERS **WHERE** ID = 2;
UPDATE table_name **SET** column1 = value1, column2 = value2, ... **WHERE** condition;	**UPDATE** CUSTOMERS **SET** Name = ' Anjali' **WHERE** ID = 4;
	UPDATE Customers **SET** ContactName = 'Juan' **WHERE** Country = 'Mexico';

SQL - GROUP BY FUNCTIONS

- Group functions are built-in SQL functions that operate on groups of rows and return one value for the entire group.
- These functions are: COUNT, MAX, MIN, AVG, SUM, DISTINCT.
  - The COUNT() function returns the number of rows that matches a specified criteria.
  - The AVG() function returns the average value of a numeric column.
  - The SUM() function returns the total sum of a numeric column.
  - The MIN() function returns the smallest value of the selected column.
  - The MAX() function returns the largest value of the selected column.

Syntax	Example
SELECT MIN(column_name) **FROM** table_name **WHERE** condition;	**SELECT** MIN(Price) **AS** SmallestPrice **FROM** PRODUCTS
SELECT MAX(column_name) **FROM** table_name **WHERE** condition;	**SELECT** MAX(Price) **AS** LargestPrice **FROM** PRODUCTS

DML – GROUP BY CLAUSE

- The GROUP BY clause groups records into summary rows.
- GROUP BY returns one records for each group.
- GROUP BY typically also involves aggregates: COUNT, MAX, SUM, AVG, etc.
- GROUP BY can group by one or more columns.

Syntax	Example
SELECT column_name(s) **FROM** table_name **WHERE** condition **GROUP BY** column_name(s) **ORDER BY** column_name(s);	
SELECT COUNT(column_name),column_name **FROM** table_name **GROUP BY** (column_name	**SELECT** COUNT(Id), ADDRESS **FROM** CUSTOMERS **GROUP** BY ADDRESS;
	SELECT NAME, SUM(SALARY) **FROM** CUSTOMERS **GROUP BY** NAME;

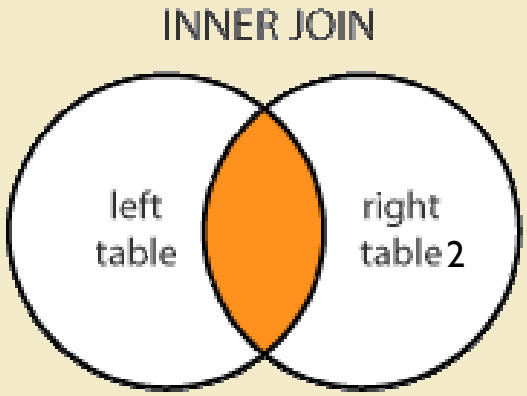
DML - ORDER BY CLAUSE

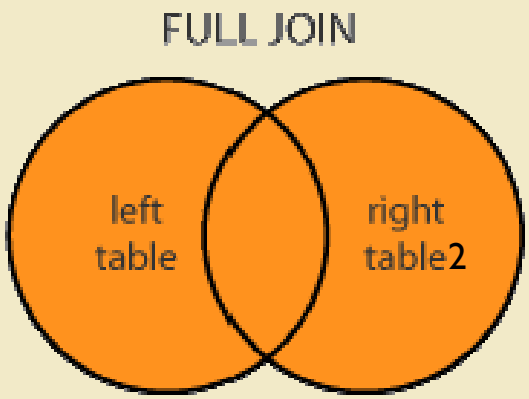
Syntax	Example
SELECT column1, column2,…columnN **FROM** table_name **WHERE** [conditions] **GROUP BY** column1, column2 **ORDER BY** column1, column2	**SELECT** COUNT(Id), ADDRESS **FROM** customers **GROUP BY** ADDRESS;
	SELECT * **FROM** CUSTOMERS **ORDER BY** NAME **,**SALARY;
	SELECT ID, NAME, SALARY **FROM** CUSTOMERS **WHERE** ID IN(1,4,7) **ORDER BY** NAME **,**SALARY;

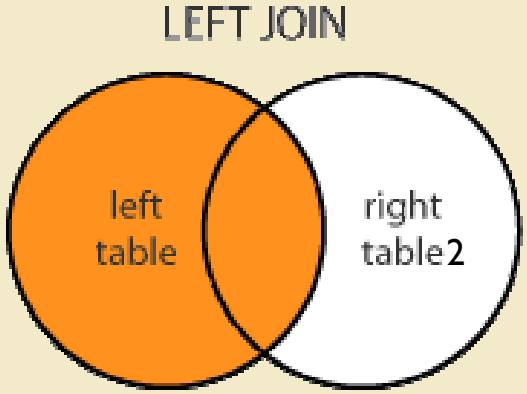
JOINS

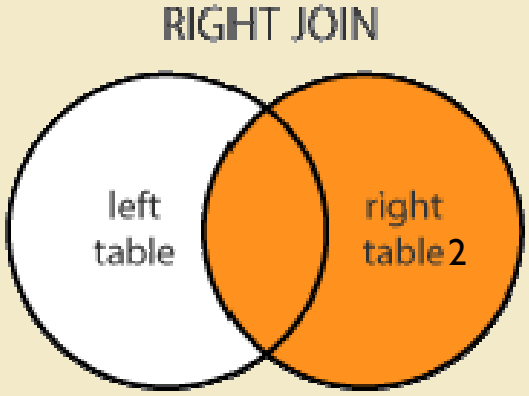
- A SQL JOIN combines records from two tables.
- A JOIN locates related column values in the two tables.
- Types of joins
  - **INNER JOIN:** Select records that have matching values in both tables.
  - **LEFT (OUTER) JOIN:** Select all records from the first (left-most) table and only matching records from right table.
  - **RIGHT (OUTER) JOIN:** Select all records from the second (right-most) table and only matching records from left table. records.
  - **FULL (OUTER) JOIN:** Selects all records that match either left or right table records.

JOINS

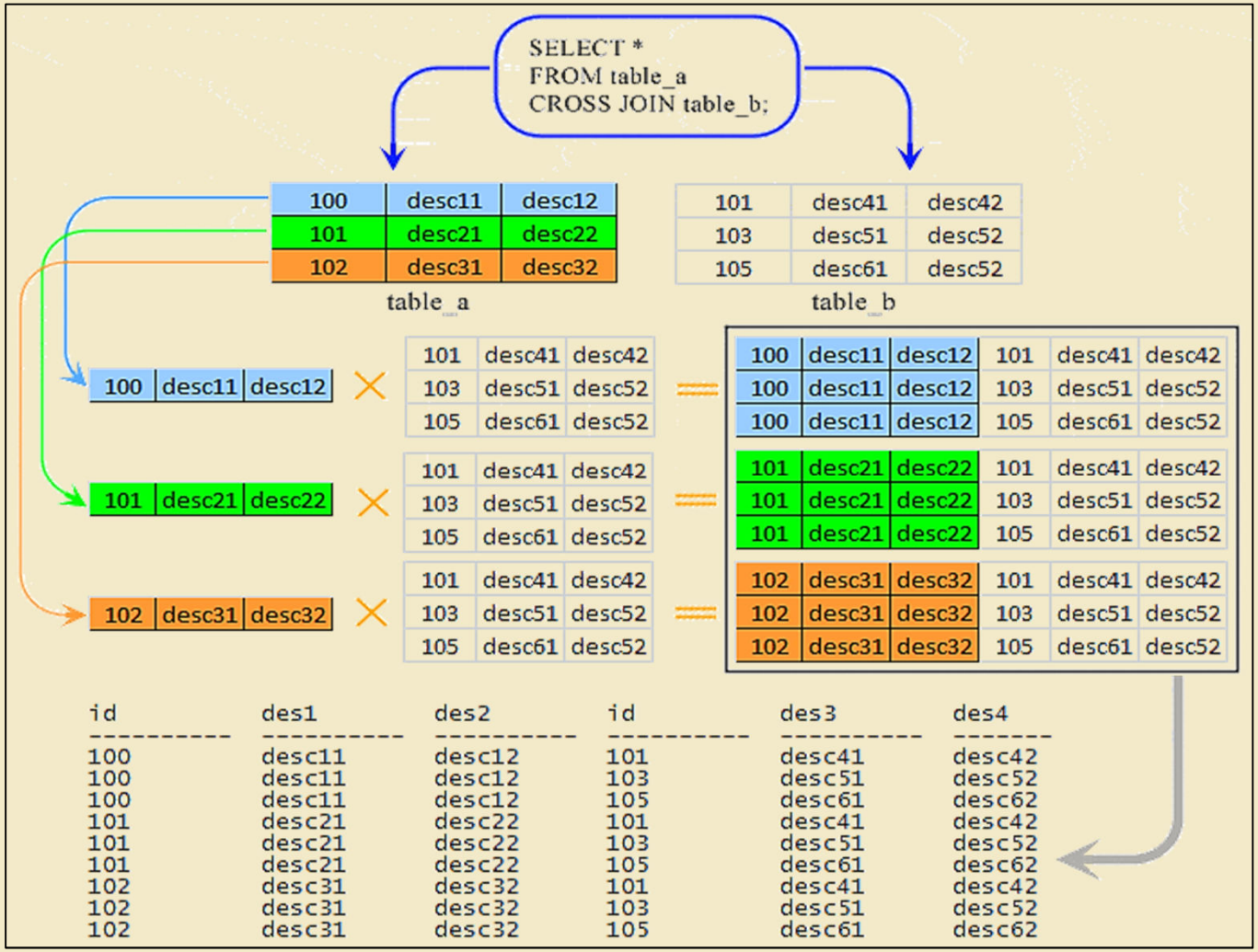

INNER JOIN
left table
right table2


FULL JOIN
left table
right table2


LEFT JOIN
left table
right table2


RIGHT JOIN
left table
right table2

CARTESIAN JOIN



DML – JOINS

Example full join (achieved by union clause)

```
select * from depositors left join customer_demo on depositors.c_name =
customer_demo.cust_name
union
select * from borrower cross join customer_demo on borrower.c_name =
customer_demo.cust_name;
```

In first select depositor and customer_demo table joined and
In second select borrower and customer_demo joined

Example – Catesian product (achieved by cross product)

```
select * from borrower cross join customer_demo;
```

Tables : borrower and customer_demo

DML – JOINS

Example – Intersection operator(achieved by IN clause)
SELECT c_name FROM depositors **WHERE c_name IN (SELECT c_name FROM borrower);** Tables : depositor and borrower
Example – Difference operator (achieved by NOT IN clause)
SELECT c_name FROM depositors **WHERE c_name NOT IN (SELECT c_name FROM borrower);**

JOIN

CREATE	INSERT
CREATE TABLE ORDERS(OID INT NOT NULL, DATE_TIME datetime, CUSTOMER_ID INT NOT NULL, AMOUNT DECIMAL (18, 2), PRIMARY KEY (OID));	**INSERT INTO** ORDERS(OID,DATE_TIME,ID,AMOUNT) **VALUES** (102,'2012-12-31 11:30:45',3 ,3000);

OID	DATE	CUSTOMER_ID	AMOUNT
102	2009-10-08 00:00:00	3	3000
100	2009-10-08 00:00:00	3	1500
101	2009-11-20 00:00:00	2	1560
103	2008-05-20 00:00:00	4	2060

DML – JOINS

Syntax
SELECT column_name(s) **FROM** table1 **INNER JOIN** table2 **ON** table1.column_name = table2.column_name;
Example
SELECT ID, NAME, AMOUNT, DATE_TIME **FROM** ORDERS **INNER JOIN** CUSTOMERS **ON** ORDERS.cust_id = CUSTOMERS.ID;

DML – JOINS

Syntax
SELECT column-names **FROM** table-name1, table-name2 **WHERE** condition [column-name1 = column-name2]
Example
SELECT ID, NAME, AGE, AMOUNT **FROM** CUSTOMERS, ORDERS **WHERE** CUSTOMERS.ID=ORDERS.CUSTOMER_ID;

SQL - CONSTRAINTS

- SQL constraints are used to specify rules for data in a table.
- Constraints are used to limit the type of data stored into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.
- Constraints can be
  - Column level : apply to a column.
  - Table level : apply to the whole table.
- The following constraints are commonly used in SQL:
  - **NOT NULL:** Ensures that a column cannot have a NULL value
  - **UNIQUE:** Ensures that all values in a column are different
  - **PRIMARY KEY:** A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
  - **FOREIGN KEY:** Uniquely identifies a row/record in another table
  - **CHECK:** Ensures that all values in a column satisfies a specific condition
  - **DEFAULT:** Sets a default value for a column when no value is specified
  - **INDEX:** Used to create and retrieve data from the database very quickly

NOT NULL CONSTRIANT

- By default, a column can hold NULL values.
- The NOT NULL constraint enforces a column to NOT accept NULL values.

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255) NOT NULL,
    Age int
);
```

UNIQUE CONSTRAINT

- The UNIQUE constraint ensures that all values in a column are different.
- Table can have many UNIQUE constraints.

Syntax	Example
CREATE TABLE <table_name> (<column_name> <data_type>, <column_name2> <data_type>, ... UNIQUE(column_name));	CREATE TABLE Persons (ID int NOT NULL, LastName varchar(255) NOT NULL, FirstName varchar(255), Age int, UNIQUE (ID));
	CREATE TABLE Persons (ID int NOT NULL, LastName varchar(255) NOT NULL, FirstName varchar(255), Age int, CONSTRAINT UC_Person UNIQUE (ID, LastName));

UNIQUE CONSTRAINT

- To create a UNIQUE constraint on the column when the table is already created.

Syntax	Example
ALTER TABLE <table_name> ADD UNIQUE (<column_name>);	ALTER TABLE Persons ADD UNIQUE (ID);
ALTER TABLE <table_name> ADD CONSTRAINT <constraint_name> UNIQUE (<column_name1>,<column_name2>,...);	ALTER TABLE Persons ADD CONSTRAINT UC_Person UNIQUE (ID,LastName);
ALTER TABLE <table_name> DROP INDEX <constraint_name>;	ALTER TABLE Persons DROP INDEX UC_Person;

PRIMARY KEY CONSTRAINT

- The PRIMARY KEY constraint uniquely identifies each record in a database table.
- Primary keys must contain UNIQUE values, and cannot contain NULL values.
- A table can have only one primary key, which may consist of single or multiple fields.

Syntax	Example
CREATE TABLE <table_name> (<column_name> <data_type>, <column_name2> <data_type>, ... PRIMARY KEY(column_name));	CREATE TABLE Persons (ID int NOT NULL, LastName varchar(255) NOT NULL, FirstName varchar(255), Age int, PRIMARY KEY (ID));
CREATE TABLE <table_name> (<column_name> <data_type>, <column_name2> <data_type>, ... CONSTRAINT <constraint_name> PRIMARY KEY(column_names));	CREATE TABLE Persons (ID int NOT NULL, LastName varchar(255) NOT NULL, FirstName varchar(255), Age int, CONSTRAINT PK_Person PRIMARY KEY (ID,LastName));

PRIMARY KEY CONSTRAINT

Syntax	Example
ALTER TABLE <table_name> ADD PRIMARY KEY (column_name);	ALTER TABLE CUSTOMER ADD PRIMARY KEY (ID);
ALTER TABLE <table_name> ADD CONSTRAINT <constraint_name> PRIMARY KEY (<column_names>);	ALTER TABLE CUSTOMERS ADD CONSTRAINT PK_CUSTID PRIMARY KEY (ID, NAME);
ALTER TABLE <table_name> DROP PRIMARY KEY ;	ALTER TABLE CUSTOMERS DROP PRIMARY KEY ;
ALTER TABLE <table_name> DROP CONSTRAINT <constraint_name>;	ALTER TABLE Persons DROP CONSTRAINT PK_Person;
	DROP PRIMARY KEY;

FOREIGN KEY CONSTRAINT

- A FOREIGN KEY is a key used to link two tables together.
- A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table.
- The table containing the foreign key is called the child table, and the table containing the candidate key is called the referenced or parent table.
- The FOREIGN KEY constraint prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the table it points to.

FOREIGN KEY CONSTRAINT

Syntax	Example

	`ALTER TABLE Orders`
	`ADD CONSTRAINT FK_PersonOrder`
	`FOREIGN KEY (PersonID) REFERENCES`
	`Persons(PersonID);`

	`ALTER TABLE Orders` `DROP CONSTRAINT FK_PersonOrder;`

FOREIGN KEY CONSTRAINT

Syntax	Example
CREATE TABLE <table_name> (<column_name> <data_type>, <column_name2> <data_type>, …. …. CONSTRAINT <FK_KEY FOREIGN KEY (column_name)); REFERENCES <Parenttable_name> (<Primarykey_parenttable>)	CREATE TABLE Orders (OrderID int NOT NULL, OrderNumber int NOT NULL, PersonID int, PRIMARY KEY (OrderID), CONSTRAINT FK_KEY FOREIGN KEY (PersonID) REFERENCES Persons(PersonID));
	CREATE TABLE borrower(c_name VARCHAR (20) NOT NULL, l_no VARCHAR (10) NOT NULL, constraint fk_borrower foreign key (c_name) references customer_demo(cust_name), foreign key (l_no) references loan(loan_no)) engine = innodb;

Thank You