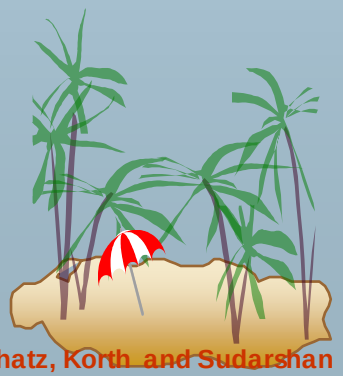
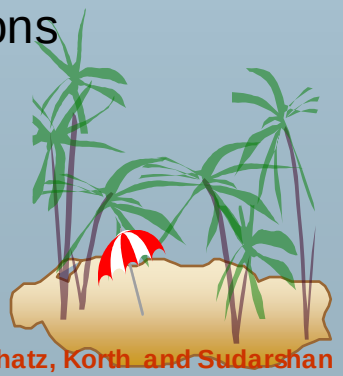
Chapter 1: Introduction

- Purpose of Database Systems
- View of Data
- Data Models
- Data Definition Language
- Data Manipulation Language
- Transaction Management
- Storage Management
- Database Administrator
- Database Users
- Overall System Structure



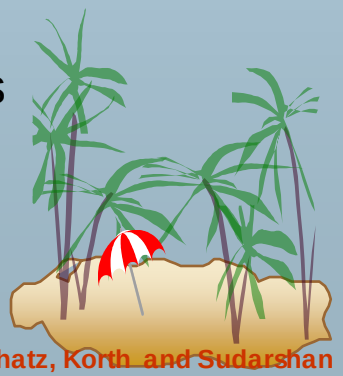
Database Management System (DBMS)

- Collection of interrelated data
- Set of programs to access the data
- DBMS contains information about a particular enterprise
- DBMS provides an environment that is both *convenient* and *efficient* to use.
- Database Applications:
  - Banking: all transactions
  - Airlines: reservations, schedules
  - Universities: registration, grades
  - Sales: customers, products, purchases
  - Manufacturing: production, inventory, orders, supply chain
  - Human resources: employee records, salaries, tax deductions
- Databases touch all aspects of our lives



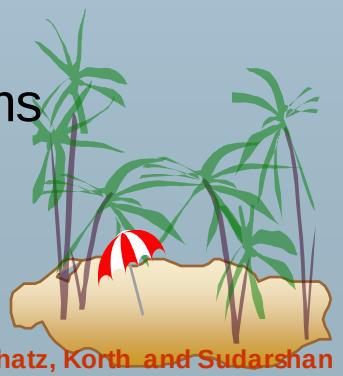
Purpose of Database System

- In the early days, database applications were built on top of file systems
- Drawbacks of using file systems to store data:
  - Data redundancy and inconsistency
    - Multiple file formats, duplication of information in different files
  - Difficulty in accessing data
    - Need to write a new program to carry out each new task
  - Data isolation — multiple files and formats
  - Integrity problems
    - Integrity constraints (e.g. account balance > 0) become part of program code
    - Hard to add new constraints or change existing ones



Purpose of Database Systems (Cont.)

- Drawbacks of using file systems (cont.)
  - Atomicity of updates
    - Failures may leave database in an inconsistent state with partial updates carried out
    - E.g. transfer of funds from one account to another should either complete or not happen at all
  - Concurrent access by multiple users
    - Concurrent accessed needed for performance
    - Uncontrolled concurrent accesses can lead to inconsistencies
      - E.g. two people reading a balance and updating it at the same time
  - Security problems
- Database systems offer solutions to all the above problems

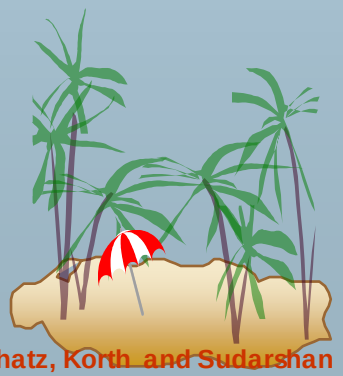


Levels of Abstraction

- Physical level describes how a record (e.g., customer) is stored.
- Logical level: describes data stored in database, and the relationships among the data.

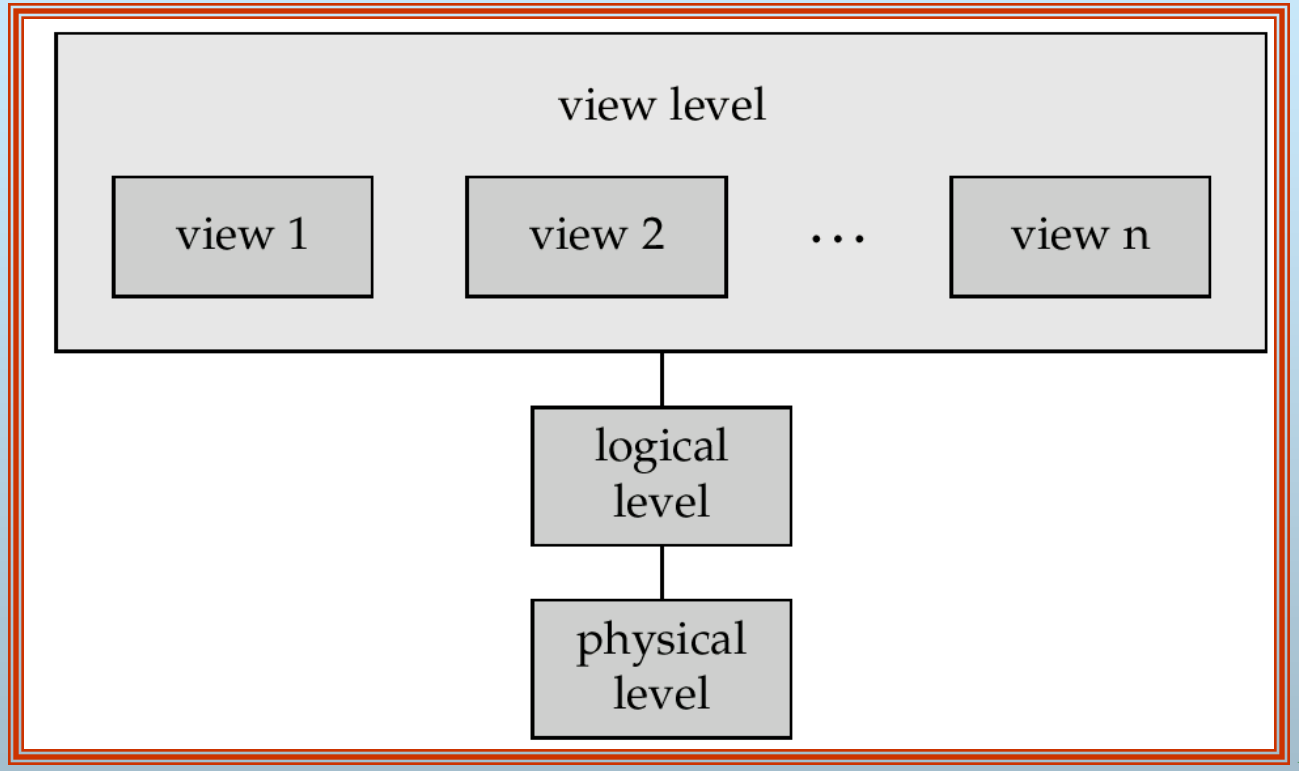
```
type customer = record
                    name : string;
                    street : string;
                    city : integer;
                end;
```

- View level: application programs hide details of data types. Views can also hide information (e.g., salary) for security purposes.



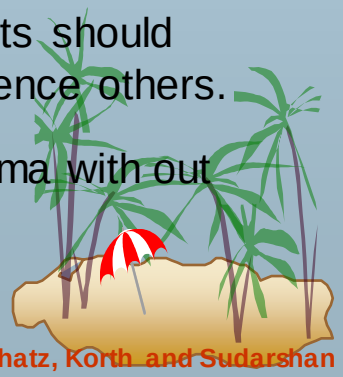
View of Data

An architecture for a database system



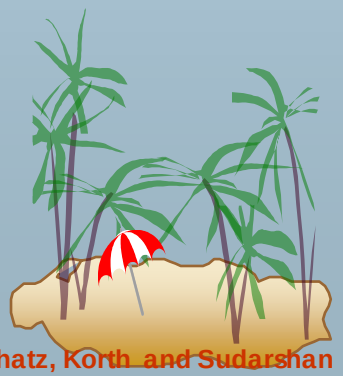
Instances and Schemas

- Similar to types and variables in programming languages
- **Schema** – the logical structure of the database
  - e.g., the database consists of information about a set of customers and accounts and the relationship between them)
  - Analogous to type information of a variable in a program
  - **Physical schema**: database design at the physical level
  - **Logical schema**: database design at the logical level
  - **Sub schemas**: database design at the view level
- **Instance** – the actual content of the database at a particular point in time
  - Analogous to the value of a variable
- **Physical Data Independence** – the ability to modify the physical schema without changing the logical schema
  - Applications depend on the logical schema
  - In general, the interfaces between the various levels and components should be well defined so that changes in some parts do not seriously influence others.
- **Logical Data Independence** – the ability to modify the logical schema without changing the sub schemas.



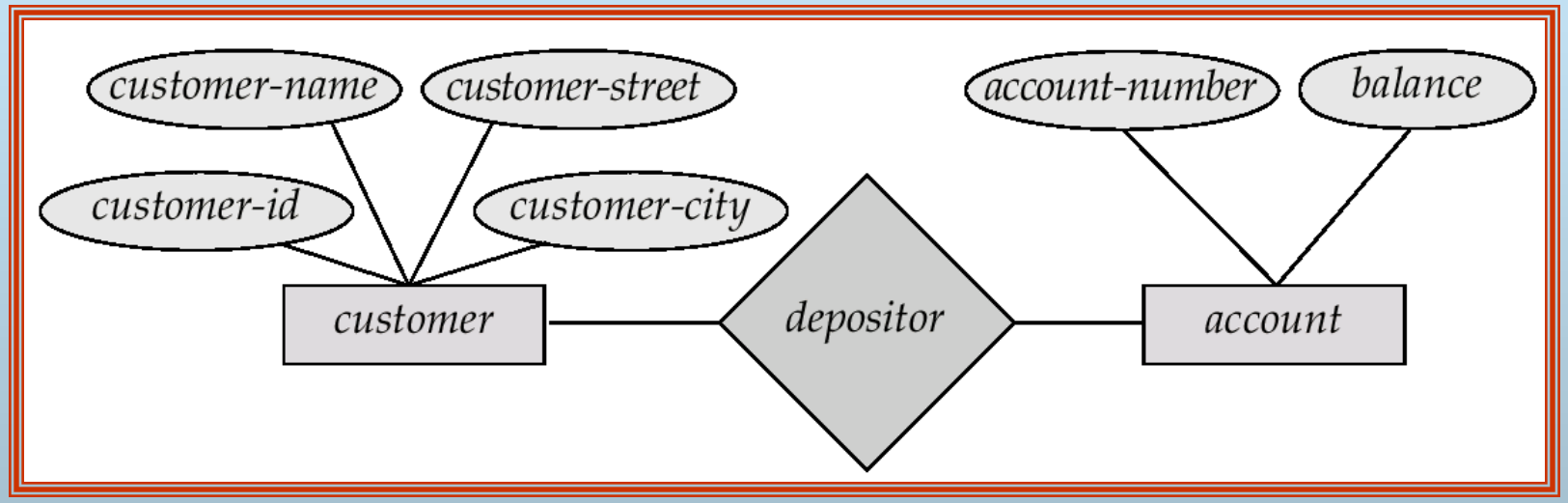
Data Models

- Data models are used to show how the data is connected and stored inside a system.
- Data models mainly represent the relationship between the data.
- A collection of tools for describing
  - data
  - data relationships
  - data semantics
  - data constraints
- Entity-Relationship model
- Relational model
- Other models:
  - object-oriented model
  - semi-structured data models
  - Older models: network model and hierarchical model



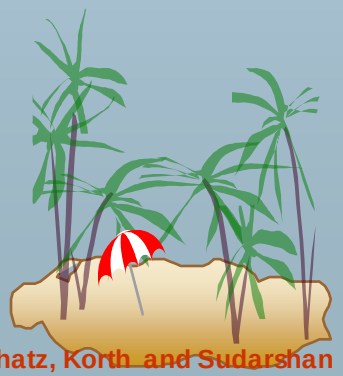
Entity-Relationship Model

Example of schema in the entity-relationship model



Entity Relationship Model (Cont.)

- E-R model of real world
  - Entities (objects)
    - E.g. customers, accounts, bank branch
  - Relationships between entities
    - E.g. Account A-101 is held by customer Johnson
    - Relationship set *depositor* associates customers with accounts
- Widely used for database design
  - Database design in E-R model usually converted to design in the relational model (coming up next) which is used for storage and processing

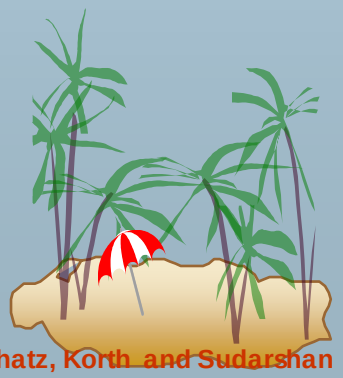


Relational Model

- Example of tabular data in the relational model

Attributes

Customer-id	*customer-name*	*customer-street*	*customer-city*	*account-number*
192-83-7465	Johnson	Alma	Palo Alto	A-101
019-28-3746	Smith	North	Rye	A-215
192-83-7465	Johnson	Alma	Palo Alto	A-201
321-12-3123	Jones	Main	Harrison	A-217
019-28-3746	Smith	North	Rye	A-201



A Sample Relational Database

customer-id	*customer-name*	*customer-street*	*customer-city*
192-83-7465	Johnson	12 Alma St.	Palo Alto
019-28-3746	Smith	4 North St.	Rye
677-89-9011	Hayes	3 Main St.	Harrison
182-73-6091	Turner	123 Putnam Ave.	Stamford
321-12-3123	Jones	100 Main St.	Harrison
336-66-9999	Lindsay	175 Park Ave.	Pittsfield
019-28-3746	Smith	72 North St.	Rye

(a) The *customer* table

account-number	*balance*
A-101	500
A-215	700
A-102	400
A-305	350
A-201	900
A-217	750
A-222	700

(b) The *account* table

customer-id	*account-number*
192-83-7465	A-101
192-83-7465	A-201
019-28-3746	A-215
677-89-9011	A-102
182-73-6091	A-305
321-12-3123	A-217
336-66-9999	A-222
019-28-3746	A-201

(c) The *depositor* table

Other models

- **Object Oriented model**:

An object-oriented data model is one of the most developed data models which contains video, graphical files, and audio.

- **Semi structured Data models:**

The semi-structured data model is a self-describing data model.

The data stored in this model is generally associated with a scheme which is contained within the data property known as self-describing property.

- **Network data model:**

Represent complex data relationships more effectively

- **Hierarchical data model:**

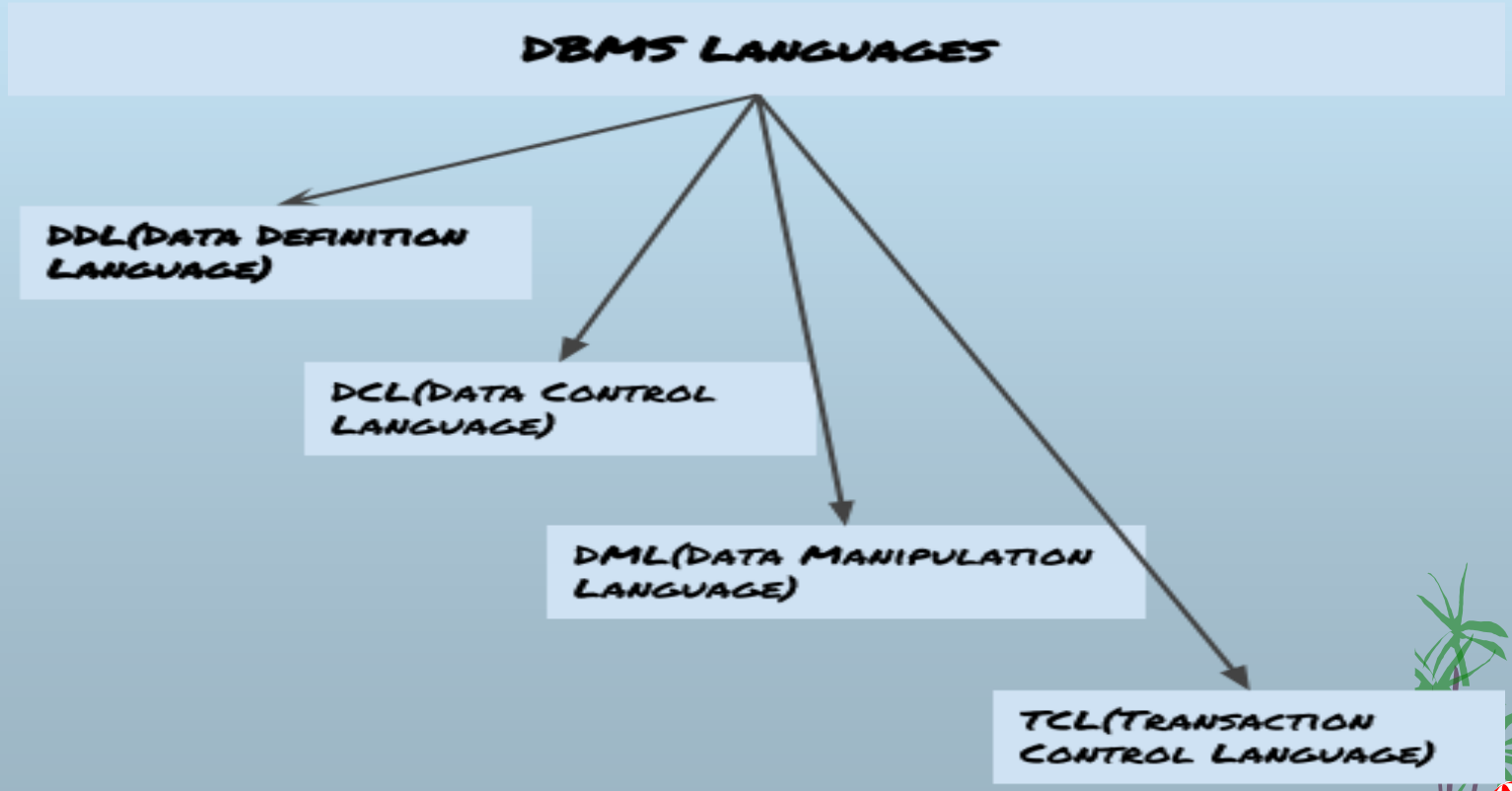
The hierarchical model is based on the parent-child hierarchical relationship.

In this model, there is one parent entity with several children entity. At the top, there should be only one entity which is called root.

For example, an organization is the parent entity called root and it has several children entities like clerk, officer and many more.

Data Base Languages

- Database languages are used to read, update and store data in a database. Ex:SQL
- Types of Database Language

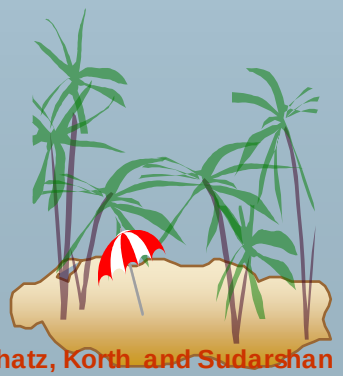


Data Definition Language (DDL)

- DDL is used for specifying the database schema. It is used for creating tables, schema, indexes, constraints etc. in database.

Lets see the operations that we can perform on database using DDL:

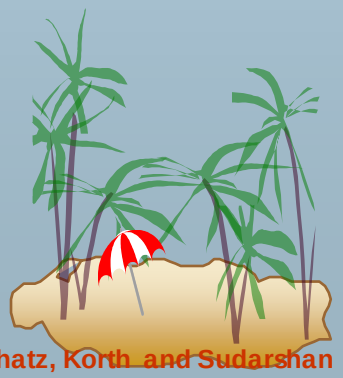
- To create the database instance – **CREATE**
- To alter the structure of database – **ALTER**
- To drop database instances – **DROP**
- To delete tables in a database instance – **TRUNCATE**
- To rename database instances – **RENAME**
- To Comment – **Comment**



Data Definition Language (DDL)

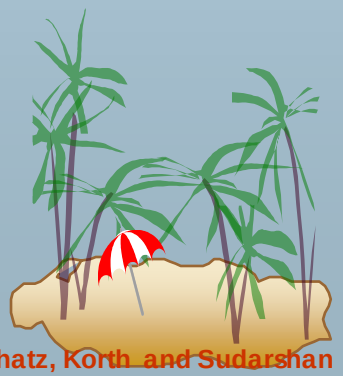
- Specification notation for defining the database schema
  - E.g.

    create table *account* (
    account-number **char**(10),
    balance **integer**)
- DDL compiler generates a set of tables stored in a *data dictionary*
- Data dictionary contains metadata (i.e., data about data)
  - database schema
  - Data *storage and definition* language
    - language in which the storage structure and access methods used by the database system are specified



Data Manipulation Language (DML)

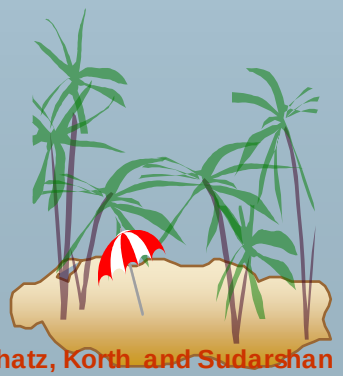
- Language for accessing and manipulating the data organized by the appropriate data model
  - DML also known as query language
  - DML is used for accessing and manipulating data in a database.
- Basically there are Two classes of languages
  - Procedural – user specifies what data is required and how to get those data ex: C,PASCAL
  - Nonprocedural – user specifies what data is required without specifying how to get those data
- SQL is the most widely used query language and it is Nonprocedural



Data Manipulation Language (DML)

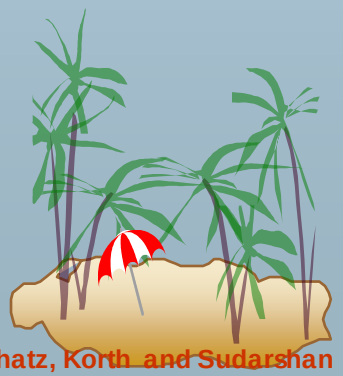
The following operations on database comes under DML:

- To read records from table(s) – **SELECT**
- To insert record(s) into the table(s) – **INSERT**
- Update the data in table(s) – **UPDATE**
- Delete all the records from the table – **DELETE**



Data Control Language (DCL)

- DCL is used for granting and revoking user access on a database –
- To grant access to user – **GRANT**
- To revoke access from user – **REVOKE**

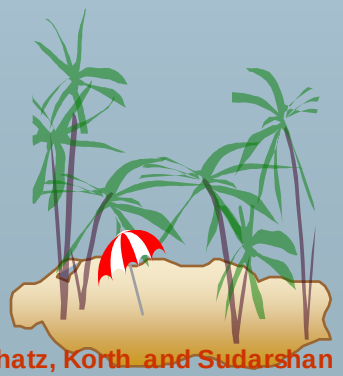


Transaction Control Language (TCL)

- The changes in the database that we made using DML commands are either performed or roll backed using TCL.
- To persist the changes made by DML commands in database or

It is used to save the transaction on the database – **COMMIT**

- To rollback the changes made to the database or It is used to restore the database to original since the last Commit – **ROLLBACK**

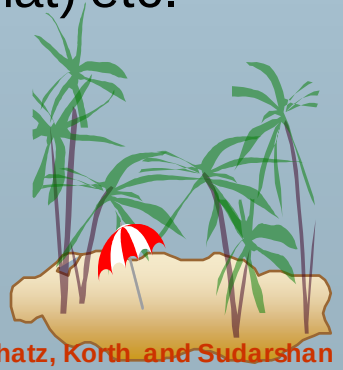


SQL

- SQL: widely used non-procedural language
- SQL is Structured Query Language, which is a computer language for storing, manipulating and retrieving data stored in relational database.
- All relational database management systems like MySQL, MS Access, Oracle, Sybase, Informix, and SQL Server use SQL as standard database language.

Also, they are using different dialects, such as:

- My SQL Server using T-SQL,
- Oracle using PL/SQL,
- MS Access version of SQL is called JET SQL (native format) etc.



SQL

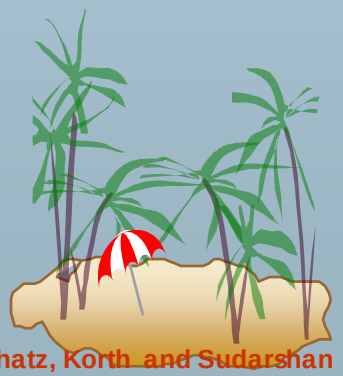
Why SQL?

- Allows users to access data in relational database management systems.
- Allows users to describe the data.
- Allows users to define the data in database and manipulate that data.
- Allows to embed within other languages using SQL modules, libraries & pre-compilers.
- Allows users to create and drop databases and tables.
- Allows users to create view, stored procedure, functions in a database.
- Allows users to set permissions on tables, procedures, and views

SQL

History:

- 1970 -- Dr. Edgar F. "Ted" Codd of IBM is known as the father of relational databases. He described a relational model for databases.
- 1974 -- Structured Query Language appeared.
- 1978 -- IBM worked to develop Codd's ideas and released a product named System/R.
- 1986 -- IBM developed the first prototype of relational database and standardized by ANSI. The first relational database was released by Relational Software and its later becoming Oracle.



SQL-Data types

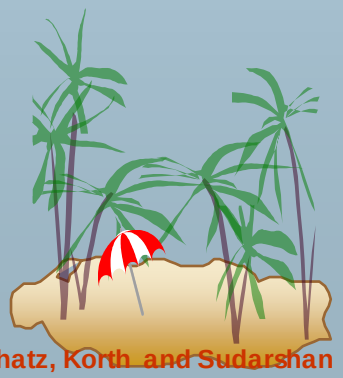
- SQL data type is an attribute that specifies type of data of any object. Each column, variable and expression has related data type in SQL.
- **Character Data types:**

Data Type Syntax	Oracle 11g	Explanation
char(size)	Maximum size of 2000 bytes.	Where ***size*** is the number of characters to store. Fixed-length strings. Space padded.
varchar2(size)	Maximum size of 4000 bytes.	Where ***size*** is the number of characters to store. Variable-length string.
long	Maximum size of 2GB.	Variable-length strings.

SQL-Data types

- **Numeric Data types:**

Data Type Syntax	Oracle 11g	Explanation
number(p,s)	Precision can range from 1 to 38. Scale can range from -84 to 127.	Where ***p*** is the precision and ***s*** is the scale. For example, number(7,2) is a number that has 5 digits before the decimal and 2 digits after the decimal.
float		
integer		
int		
smallint		



SQL-Data types

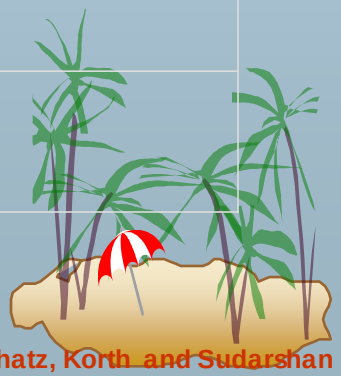
- **Date/Time Data types:**

Data Type Syntax	Oracle 11g	Explanation
date	A date between Jan 1, 4712 BC and Dec 31, 9999 AD.	

- **Large Object (LOB) Data types:**

Large Object (LOB) data types refer to large data items such as text, graphic images, video clips, and sound waveforms.

Data Type	Description	Size
BLOB	Used to store large binary objects in the database.	8 to 128 terabytes (TB)
CLOB	Used to store large blocks of character data in the database.	8 to 128 TB



SQL-DDL commands

- ***1)Creating a Database:***

Syntax:

create database *database-name*;

Example:

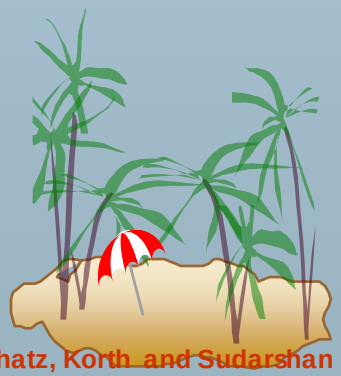
- create database Test;
- ***2)Creating a Table:***

Syntax:

create table *table-name(column-name1* datatype1,*column-name2* datatype2,*column-name3* datatype3,*column-name4* datatype4);

Example:

create table Student(id int, name varchar2(20), age int);

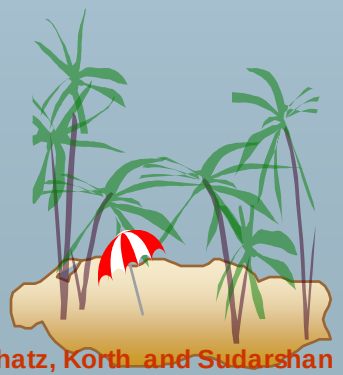


SQL-DDL commands

- **2)alter command:**

alter command is used for alteration of table structures. There are various uses of *alter* command, such as,

- to add a column to existing table
- to rename any existing column
- to change datatype of any column or to modify its size.
- *alter* is also used to drop a column.



SQL-DDL commands

- ***To Add Column to existing Table***

Syntax:

- alter table *table-name* add(column-name *datatype*);

Example :

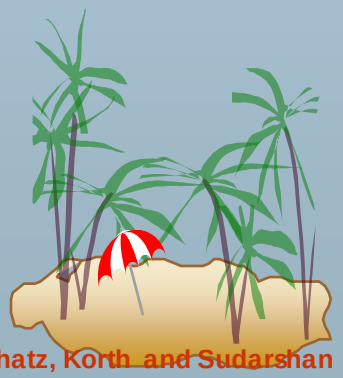
- alter table Student add(address char);
- ***To Add column with Default Value***

Syntax:

- alter table *table-name* add(column-name1 *datatype1* default *data*);

Example:

- alter table Student add(dob date default '1-Jan-99');



SQL-DDL commands

- ***To Modify an existing Column***

Syntax:

- alter table *table-name* modify(column-name *datatype*);

Example :

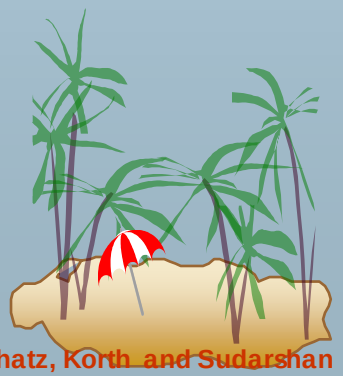
- alter table Student modify(address varchar(30));
- ***To Rename a column***

Syntax:

- alter table *table-name* rename old-column-name to column-name;

Example:

- alter table Student rename address to Location;



SQL-DDL commands

- ***To Drop a Column***

Syntax:

- alter table *table-name* drop(column-name);

Example:

- alter table Student drop(address);

SQL-DDL commands

- **3) *truncate command:***

Syntax:

- truncate table *table-name;*

Example :

- truncate table Student;
- ***4)drop command:***

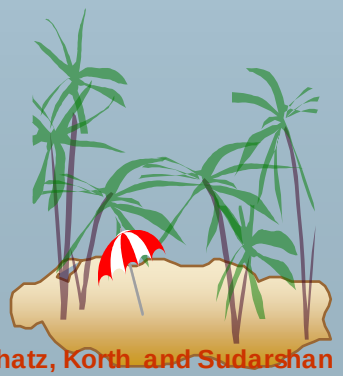
Syntax:

drop table *table-name*

Example :

drop table Student;

drop database Test;



SQL-DDL commands

- ***5)rename query:***

Syntax:

- rename table *old-table-name* to *new-table-name*

Example:

- rename table Student to Student-record;
- **6) Comment Query:**

Syntax:

-- *comment goes here or /* comment goes here */*

Example:

SELECT websites.site_name /* Author: ramu */ FROM websites;

SELECT websites.site_name - -Author:ramu FROM websites;

SQL-DML commands

- ***1) INSERT command***
- Insert command is used to insert data into a table.

Syntax:

- INSERT into *table-name* values(data1,data2,..);

Example:

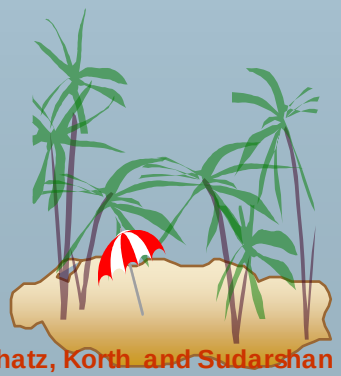
INSERT into Student values(101,'Adam',15);

Example to Insert NULL value to a column

- INSERT into Student(id,name) values(102,'Alex'); Or,
- INSERT into Student values(102,'Alex',null);

Example to Insert Default value to a column

- INSERT into Student values(103,'Chris',default)



SQL-DML commands

- ***2) UPDATE command***
- Update command is used to update a row of a table.

Syntax:

- UPDATE *table-name* set column-name = value *where* condition;

Example:

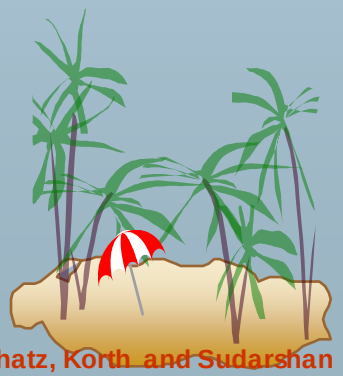
- update Student set age=18 where s_id=102;

Example to Update multiple columns

- UPDATE Student set s_name='Abhi',age=17 where s_id=103;
- ***3) Delete command***
- Delete command is used to delete data from a table. Delete command can also be used with condition to delete a particular row.

Syntax:

- DELETE from *table-name*;



SQL-DML commands

Example to Delete all Records from a Table

- DELETE from Student;

Example to Delete a particular Record from a Table

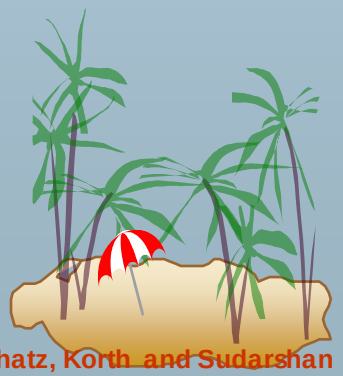
- DELETE from Student where s_id=103;
- **4) SELECT Command**
- Select query is used to retrieve data from a tables, We can retrieve complete tables, or partial by mentioning conditions using WHERE clause.

Syntax:

- SELECT column-name1, column-name2, column-name3, column-nameN from *table-name*;

Example:

- SELECT s_id, s_name, age from Student;



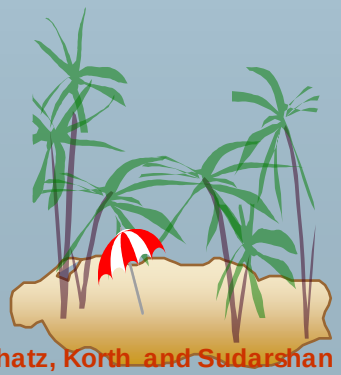
SQL-DML commands

Example to Select all Records from Table

- SELECT * from student;

Example to Select particular Record based on Condition

- SELECT * from Student WHERE s_name = 'Abhi';



SQL-TCL commands

- ***1)Commit command***
- Commit command is used to permanently save any transaction into database.

Syntax:

- *commit*;
- ***2) Rollback command***
- This command restores the database to last committed state. It is also use with save point command to jump to a savepoint in a transaction.

Syntax:

- rollback to *savepoint-name*;
- **3) *Savepoint command***
- savepoint command is used to temporarily save a transaction so that you can rollback to that point whenever necessary.

Syntax:

- Savepoint *savepoint-name*;

SQL-DCL commands

- **1) Grant Command**
- Grant Command is used for offering access or privileges to the users on the objects of the database.

Syntax:

- **GRANT privilege_name**

ON object_name

TO {user_name | PUBLIC }

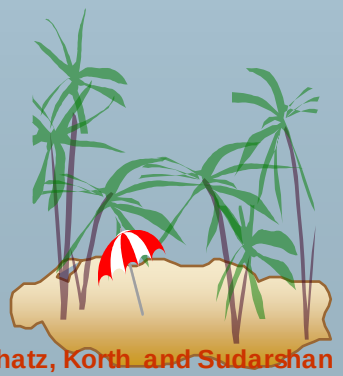
[WITH GRANT OPTION];

Example:

- GRANT ALL ON workers

TO MNO;

[WITH GRANT OPTION]



SQL-DCL commands

- **2) Revoke Command**
- revoke command is canceling the previously denied or granted permissions.

Syntax:

- **REVOKE<privilege list>**

ON <relation name or view name>

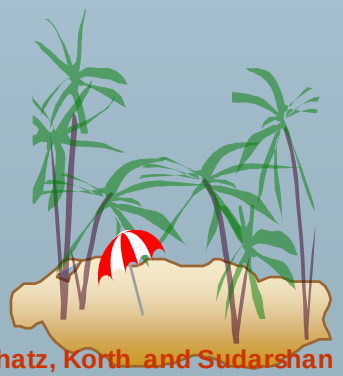
From <user name>

Example

- REVOKE UPDATE

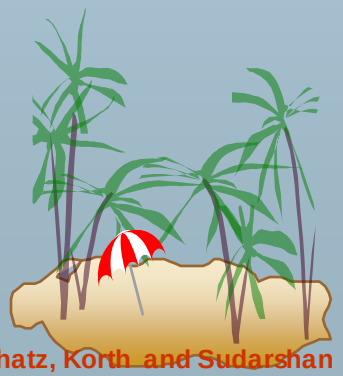
ON worker

FROM MNO;



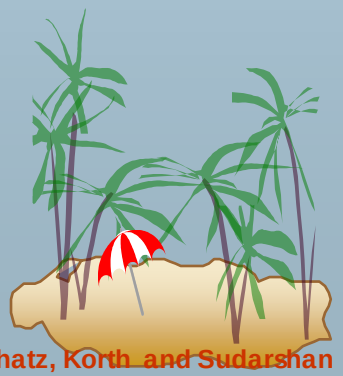
Database Users

- Users are differentiated by the way they expect to interact with the system
- Application programmers – interact with system through DML calls
- Sophisticated users – form requests in a database query language
- Specialized users – write specialized database applications that do not fit into the traditional data processing framework
- Naïve users – invoke one of the permanent application programs that have been written previously
  - E.g. people accessing database over the web, bank tellers, clerical staff

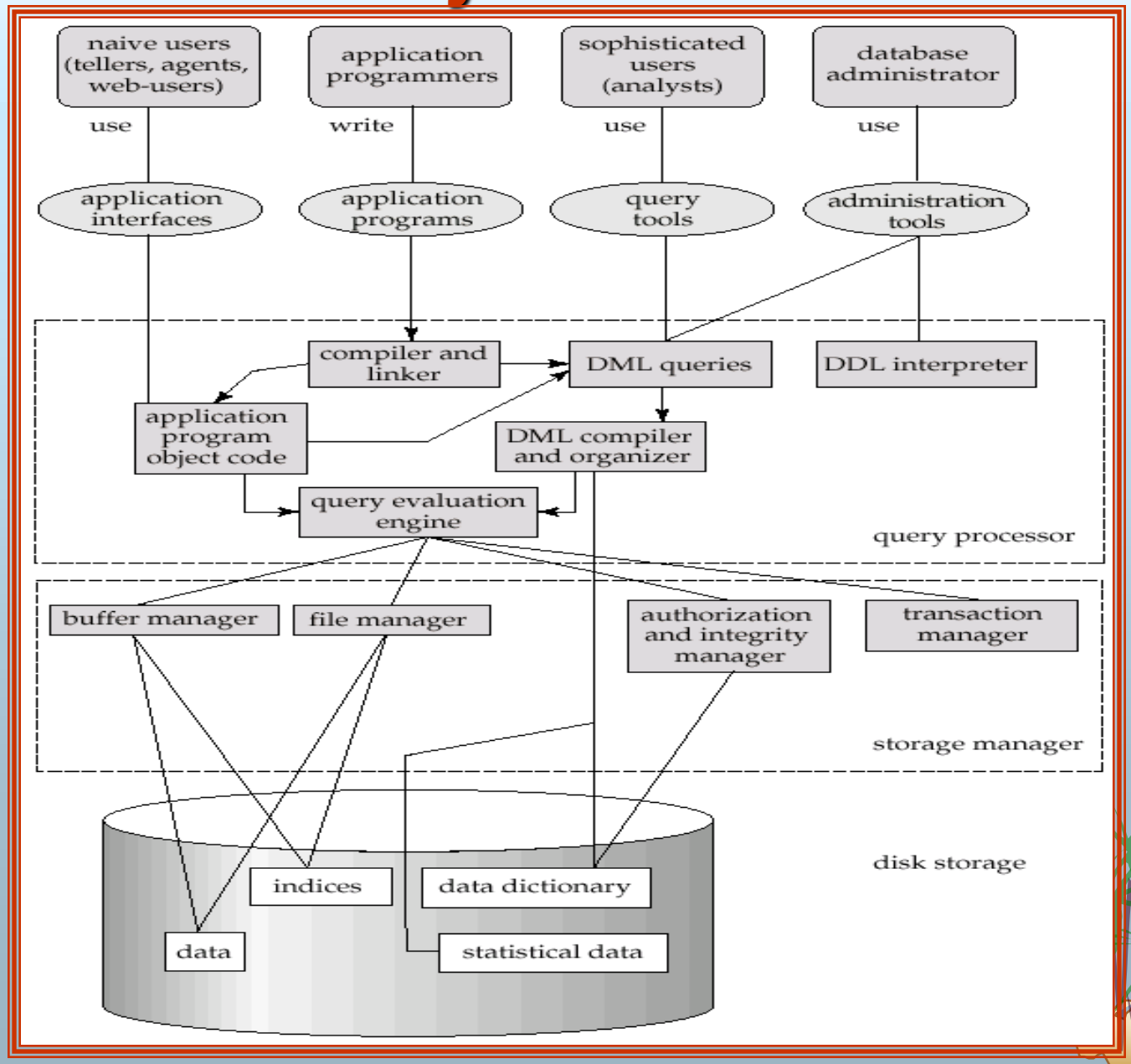


Database Administrator

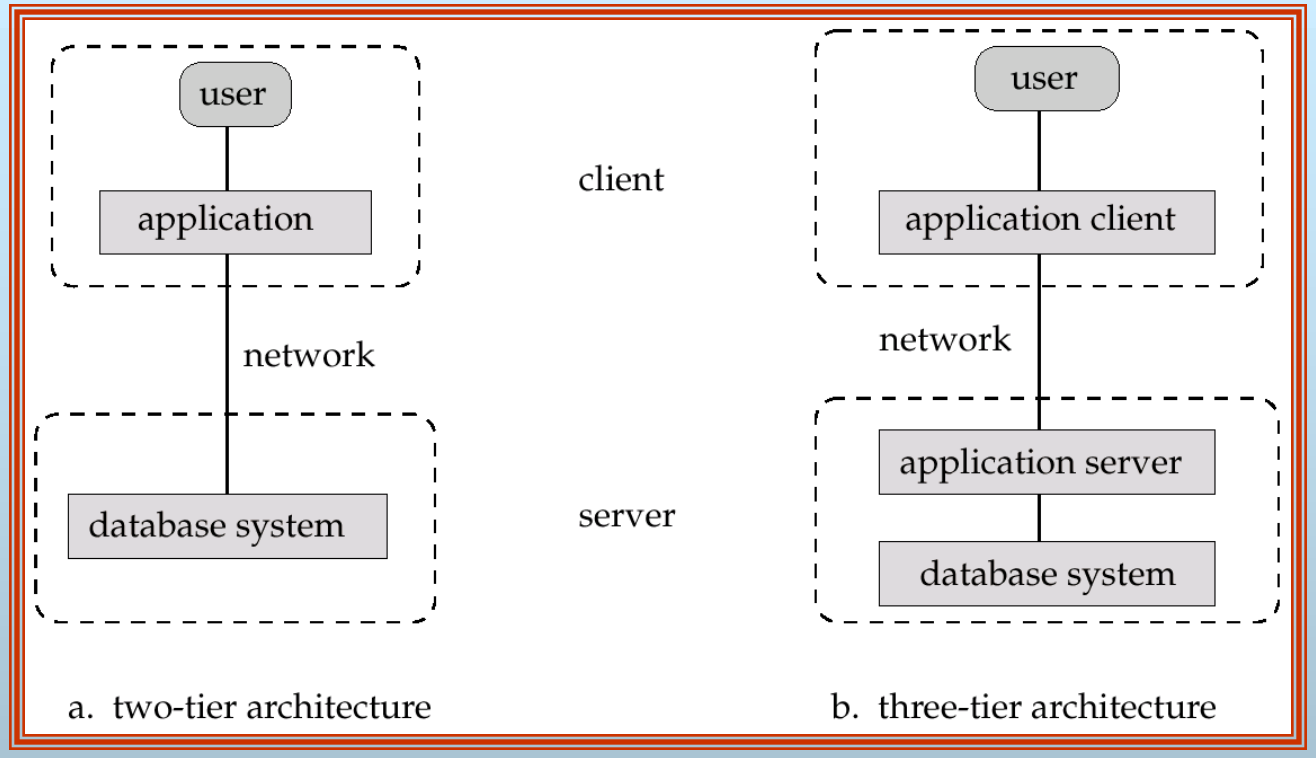
- Coordinates all the activities of the database system; the database administrator has a good understanding of the enterprise's information resources and needs.
- Database administrator's duties include:
  - Schema definition
  - Storage structure and access method definition
  - Schema and physical organization modification
  - Granting user authority to access the database
  - Specifying integrity constraints
  - Acting as liaison with users
  - Monitoring performance and responding to changes in requirements



Overall System Structure



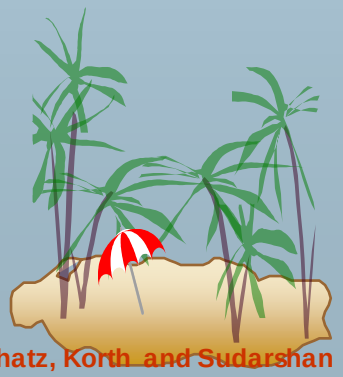
Application Architectures



- **Two-tier architecture**: E.g. client programs using ODBC/JDBC to communicate with a database
- **Three-tier architecture**: E.g. web-based applications, and applications built using "middleware"

SQL Alias

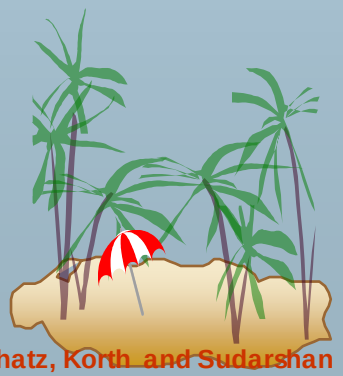
- SQL Aliases are defined for columns and tables. Basically aliases is created to make the column selected more readable.
- **For Example:** To select the first name of all the students, the query would be like:
- **Aliases for columns :**
- SELECT first_name AS Name FROM student_details ;
  or
  SELECT first_name Name FROM student_details;
- **Aliases for tables:**
- SELECT s.first_name FROM student_details s;



SQL WHERE Clause

- The WHERE Clause is used when you want to retrieve specific information from a table excluding other irrelevant data.
- **Syntax for a WHERE clause with Select statement is:**

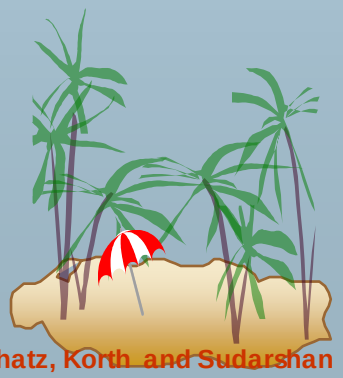
  SELECT *column_list* FROM *table-name* WHERE *condition*;
- **For Example:** To find the name of a student with id 100, the query would be like:
- SELECT first_name, last_name FROM student_details WHERE id = 100;



SQL Operators

- There are two type of Operators,namely Comparison Operators and Logical Operators.
- These operators are used mainly in the WHERE clause, HAVING clause to filter the data to be selected.
- **1) Comparison Operators:**
- Comparison operators are used to compare the column data with specific values in a condition.
- Comparison Operators are also used along with the SELECT statement to filter data based on specific conditions.

Comparison Operators	Description
=	equal to
<>, !=	is not equal to
<	less than
>	greater than
>=	greater than or equal to
<=	less than or equal to



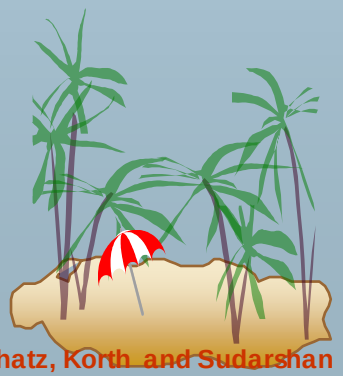
SQL Operators

- **2) Logical Operators:**

Logical Operators	Description
OR	For the row to be selected at least one of the conditions must be true.
AND	For a row to be selected all the specified conditions must be true.
NOT	For a row to be selected the specified condition must be false.

1)"OR" Logical Operator:

- If you want to select rows that satisfy at least one of the given conditions, you can use the logical operator, OR.
- **For example:** if you want to find the names of students who are studying either Maths or Science, the query would be like,
- SELECT first_name, last_name, subject
  FROM student_details
  WHERE subject = 'Maths' OR subject = 'Science';



SQL-Logical Operators

2)"AND" Logical Operator:

- If you want to select rows that must satisfy all the given conditions, you can use the logical operator, AND.
- **For Example:** To find the names of the students between the age 10 to 15 years, the query would be like:
- SELECT first_name, last_name, age
  FROM student_details
  WHERE age >= 10 AND age <= 15;

3)"NOT" Logical Operator:

- If you want to find rows that do not satisfy a condition, you can use the logical operator, NOT.
- **For example:** If you want to find out the names of the students who do not play football, the query would be like:
- SELECT first_name, last_name, games
  FROM student_details
  WHERE NOT games = 'Football'

SQL- Comparision Operators

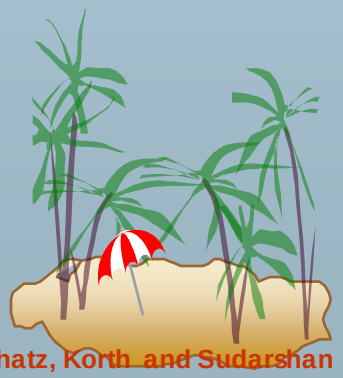
- **SQL Comparison Keywords**

1) SQL IN Operator:

- The IN operator is used when you want to compare a column with more than one value. It is similar to an OR condition.

SYNTAX:

- SELECT Column(s) FROM table_name WHERE column IN (value1, value2, ... valueN);
- **For example:** If you want to find the names of students who are studying either Maths or Science, the query would be like,
- SELECT first_name, last_name, subject
  FROM student_details
  WHERE subject IN ('Maths', 'Science');



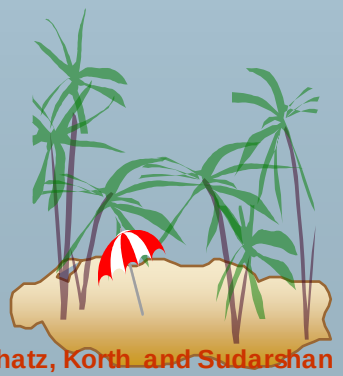
SQL-Comparision Operators

2) SQL NOT IN:

- SQL NOT IN operator is used to filter the result if the values that are mentioned as part of the IN operator is not satisfied.

Syntax:

- SELECT Column(s) FROM table_name WHERE Column NOT IN (value1, value2... valueN);
- **For example:** If you want to find the names of students who are not studying either Maths or Science, the query would be like,
- SELECT first_name, last_name, subject
  FROM student_details
  WHERE subject NOTIN ('Maths', 'Science');



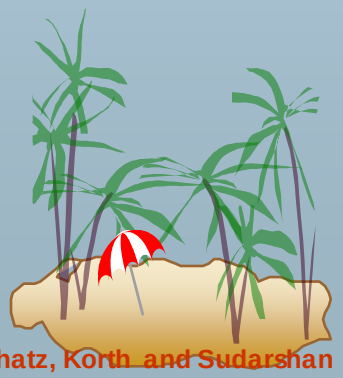
SQL-Comparision Operators

3)SQL BETWEEN … AND Operator:

- The operator BETWEEN and AND, are used to compare data for a range of values.

Syntax:

- SELECT *column_name(s)*
  FROM *table_name*
  WHERE *column_name* BETWEEN *value1* AND *value2;*
- **For Example:** to find the names of the students between age 10 to 15 years, the query would be like,
- SELECT first_name, last_name, age
  FROM student_details
  WHERE age BETWEEN 10 AND 15;



SQL-Comparision Operators

4)The SQL LIKE Operator:

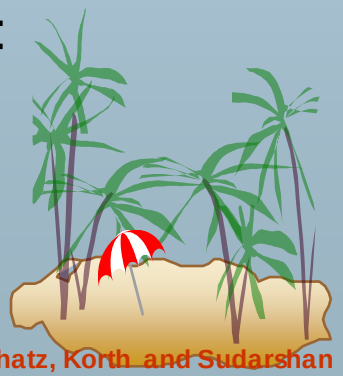
- The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.
- There are two wildcards often used in conjunction with the LIKE operator:
- % - The percent sign represents zero, one, or multiple characters
- _ - The underscore represents a single character

Syntax:

SELECT *column1, column2, ...* FROM *table_name*
WHERE *column* LIKE *pattern*;

Example:

- selects all customers with a CustomerName starting with "a":
- SELECT * FROM Customers
  WHERE CustomerName LIKE 'a%';



SQL-Like operator Examples

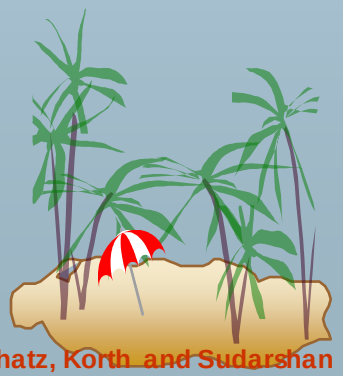
LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

SQL- Order By Clause

- Order by clause is used with Select statement for arranging retrieved data in sorted order.

Syntax:

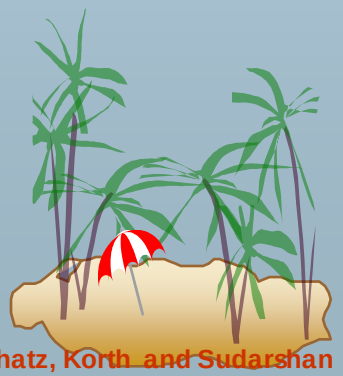
- *SELECT* column-list (or) * from table-name order by *asc|desc*;



SQL: DISTINCT clause

- DISTINCT clause is used to eliminate the duplicate values from the table.
- **Syntax:**
- SELECT DISTINCT COLUMN_NAME FROM RELATION_NAME;

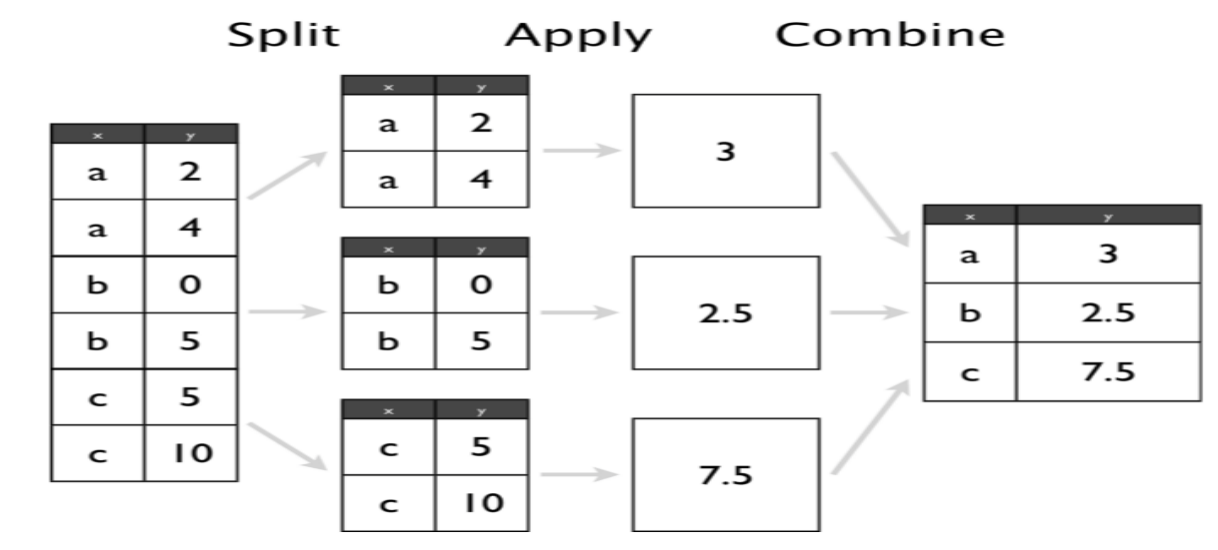
Example:
SELECT DISTINCT city FROM Employee;



SQL- Group By Clause

- The GROUP BY Clause is utilized in SQL with the SELECT statement to organize similar data into groups.
- It combines the multiple records in single or more columns using some functions.
- Generally, these functions are aggregate functions such as min(),max(),avg(), count(), and sum() to combine into single or multiple columns.
- It uses the **split-apply-combine** strategy for data analysis.

SQL- Group By Clause



Syntax:

- SELECT column_name, function(column_name) FROM table_name WHERE condition GROUP BY column_name;

Example

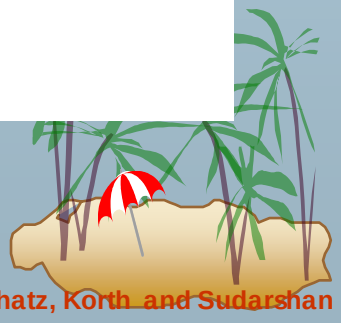
Employee

EmployeeID	Ename	DeptID	Salary
1001	John	2	4000
1002	Anna	1	3500
1003	James	1	2500
1004	David	2	5000
1005	Mark	2	3000
1006	Steve	3	4500
1007	Alice	3	3500

SELECT *DeptID, AVG(Salary)*
FROM *Employee*
GROUP BY *DeptID;*

GROUP BY Employee Table using DeptID

DeptID	AVG(Salary)
1	3000.00
2	4000.00
3	4250.00



SQL-HAVING Clause

- HAVING Clause utilized in SQL as a conditional Clause with GROUP BY Clause
- It is used to mention condition in Group based SQL functions, just like WHERE clause.
- HAVING Clause always utilized in combination with GROUP BY Clause.
- HAVING Clause restricts the data on the group records rather than individual records.
- WHERE and HAVING can be used in a single query.

Syntax:

- Select column_name, function(column_name) FROM table_name WHERE column_name condition GROUP BY column_name HAVING function(column_name) condition;

Example

Employee

EmployeeID	Ename	DeptID	Salary
1001	John	2	4000
1002	Anna	1	3500
1003	James	1	2500
1004	David	2	5000
1005	Mark	2	3000
1006	Steve	3	4500
1007	Alice	3	3500

```
SELECT DeptID, AVG(Salary)
FROM Employee
GROUP BY DeptID;
```

GROUP BY Employee Table using DeptID

DeptID	AVG(Salary)
1	3000.00
2	4000.00
3	4250.00

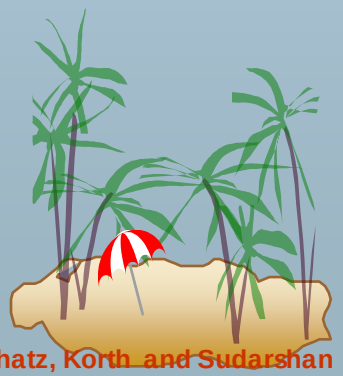
```
SELECT DeptID, AVG(Salary)
FROM Employee
GROUP BY DeptID
HAVING AVG(Salary) > 3000;
```

HAVING

DeptID	AVG(Salary)
2	4000.00
3	4250.00

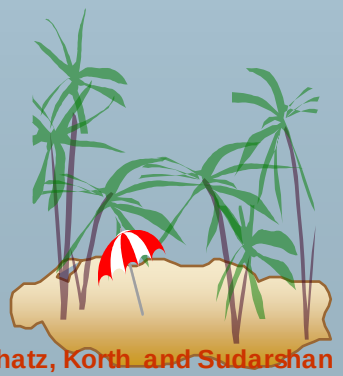
SQL: Aggregate functions

- Aggregate functions perform a calculation on a set of values and return a single value.
- Aggregate functions ignore NULL values except COUNT.
- It is used with the GROUP BY clause of the SELECT statement.
- **Following are the Aggregate functions:**
  1. AVG
  2. MAX
  3. MIN
  4. SUM
  5. COUNT()
  6. COUNT(*)



Example : <Employee> Table

Eid	Ename	Age	City	Salary
E001	ABC	29	Pune	20000
E002	PQR	30	Pune	30000
E003	LMN	25	Mumbai	5000
E004	XYZ	24	Mumbai	4000
E005	STU	32	Bangalore	25000

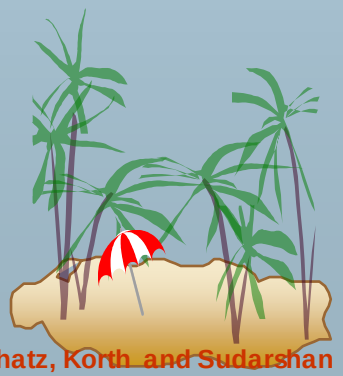


Syntax and Example

Aggregate Functions	Description	Syntax	Example	Output
AVG	It returns the average of the data values.	SELECT AVG <column_name> FROM <table_name>;	SELECT AVG(Salary) FROM Employee;	AVG(Salary) 16800
MAX	It returns the maximum value for a column.	SELECT MAX <column_name> FROM <table_name>;	SELECT MAX(Salary) FROM Employee;	MAX(Salary) 30000
MIN	It returns the minimum value for a column.	SELECT MIN <column_name> FROM <table_name>;	SELECT MIN(Salary) FROM Employee;	MIN(Salary) 4000
SUM	It returns the sum (addition) of the data values.	SELECT SUM <column_name> FROM <table_name>;	SELECT SUM(Salary) FROM Employee WHERE City='Pune';	SUM(Salary) 50000
COUNT()	It returns total number of values in a given column.	SELECT COUNT <column_name> FROM <table_name>;	SELECT COUNT(Empid) FROM Employee;	COUNT(Empid) 5
COUNT(*)	It returns the number of rows in a table.	SELECT COUNT(*) FROM <table_name>;	SELECT COUNT(*) FROM Employee;	COUNT(*) 5

SQL: Basic Form of SQL query

- The basic form of an SQL query is:
- **SELECT [DISTINCT] {*| column_name (, column_name,…)} FROM table_name [alias] (, table_name,…) [WHERE condition] [GROUP BY column_list] [HAVING condition] [ORDER BY column_list].**

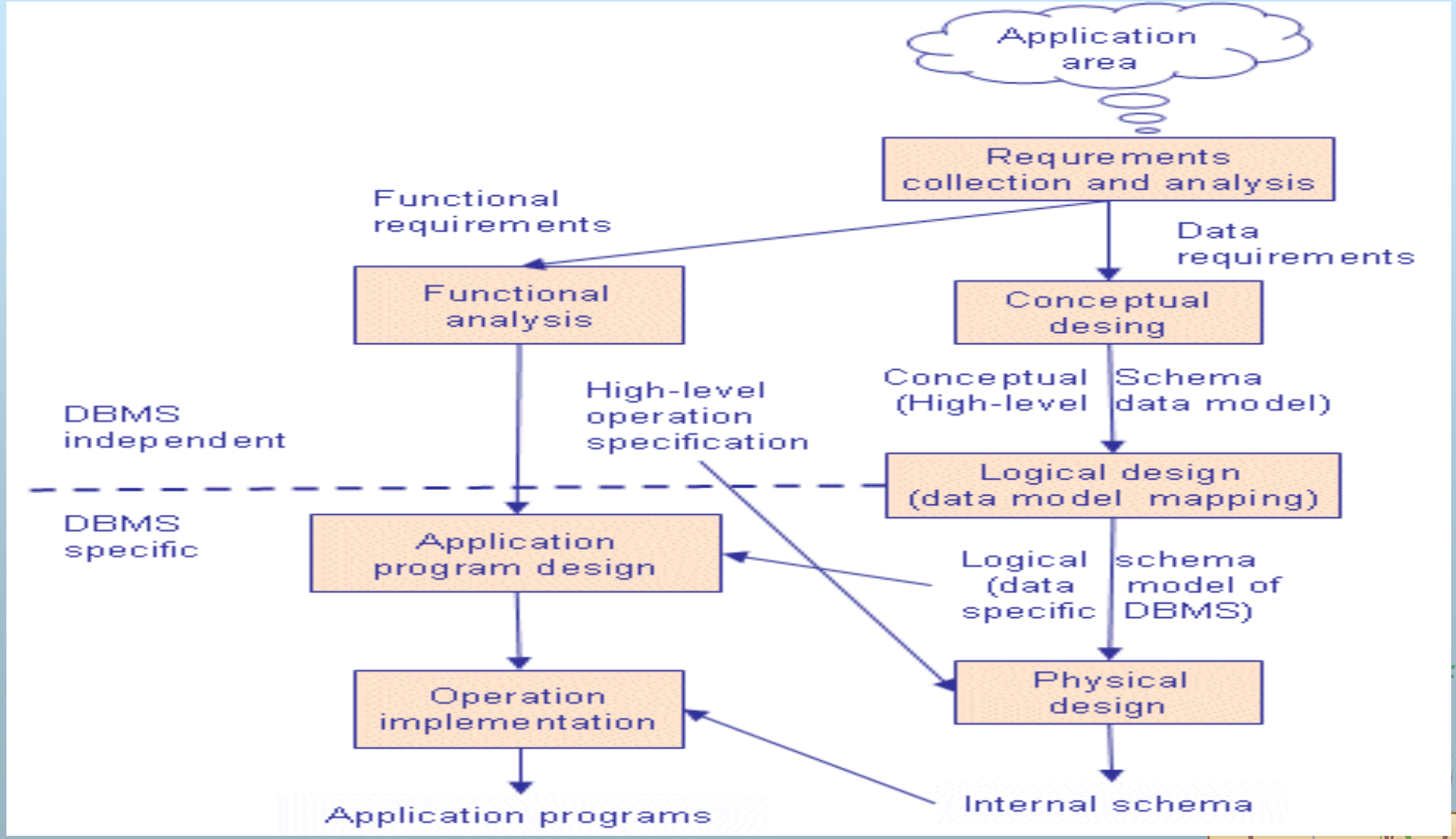


UNIT-II

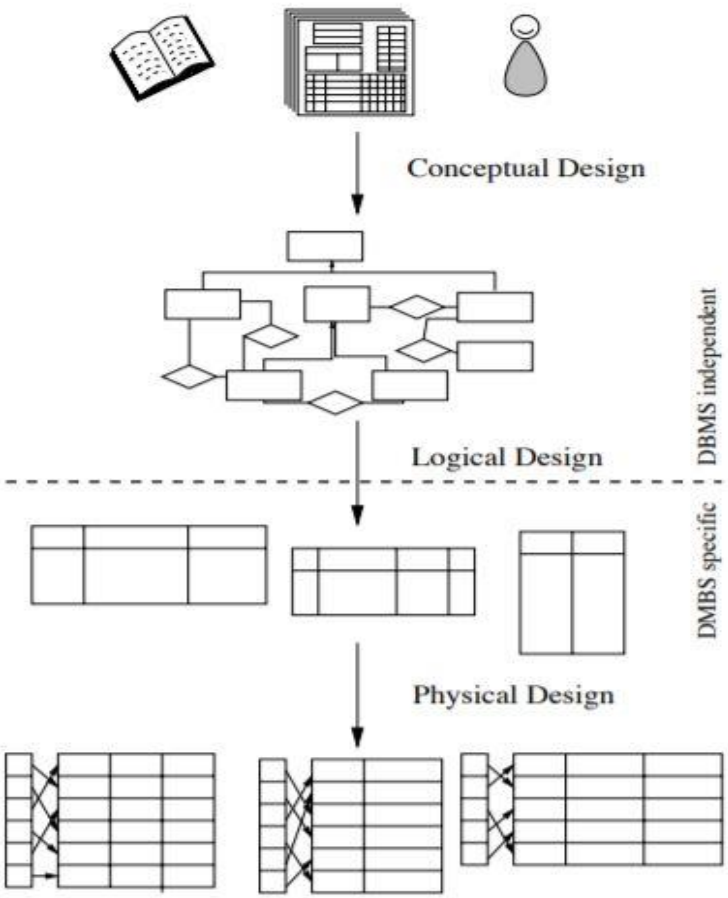
- INTRODUCTION
- DATABASE DESIGN
- INTRODUCTION TO E-R DIAGRAMS
- EXTENDED E-R FEATURES

INTRODUCTION TO DATA BASE DESIGN

- Main phases of Data base design

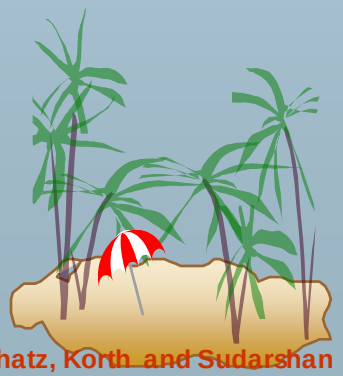


Database Design

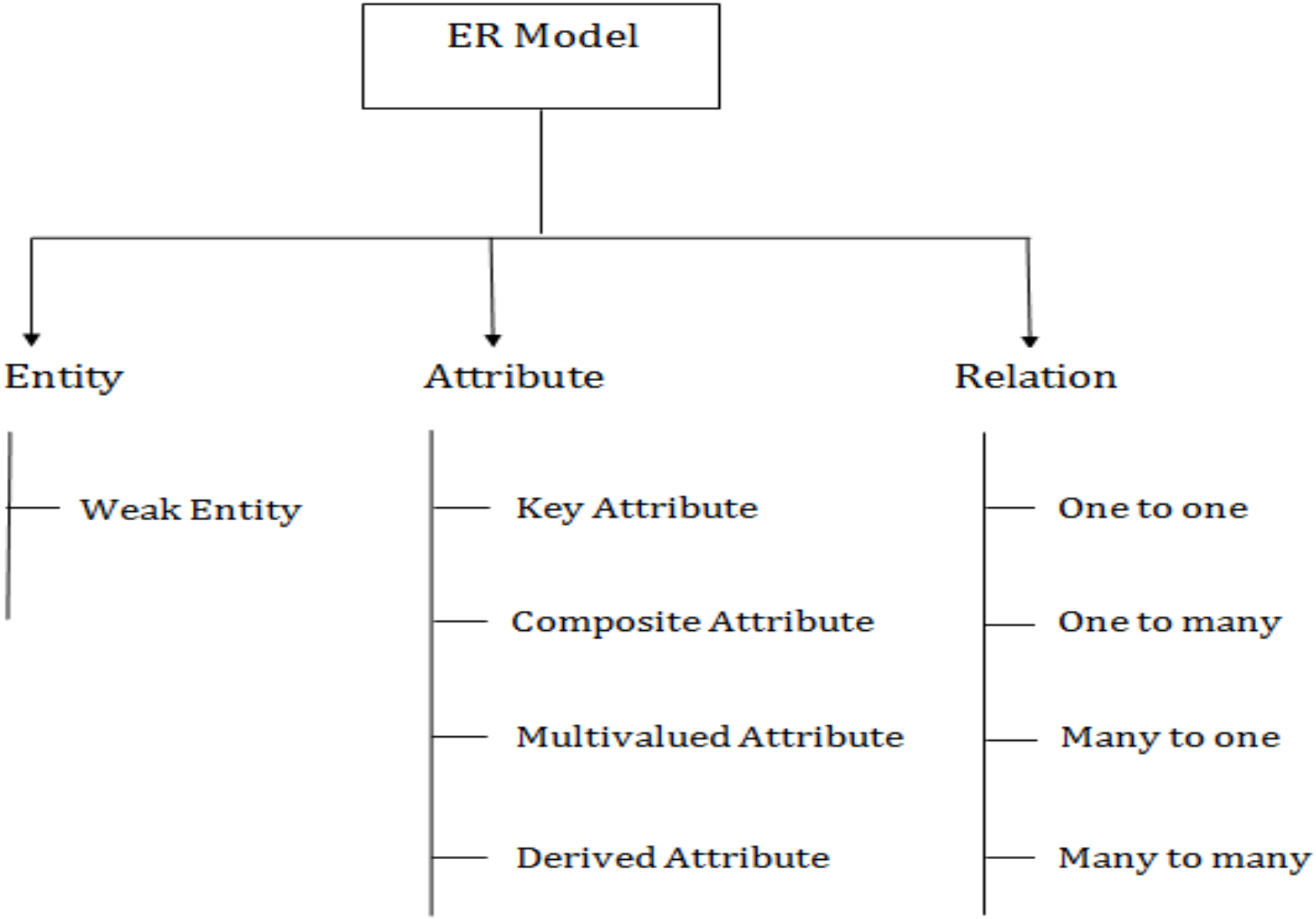


Conceptual data base design: E-R model

- **The entity-relationship model (or ER model) is a way of graphically representing the logical relationships of entities (or objects) in order to create a database.**
- **The ER model was first proposed by Peter Pin-Shan Chen of Massachusetts Institute of Technology (MIT) in the 1970s.**
- The ER model defines the conceptual view of a database. It works around real-world entities and the associations among them.
- At view level, the ER model is considered a good option for designing databases.



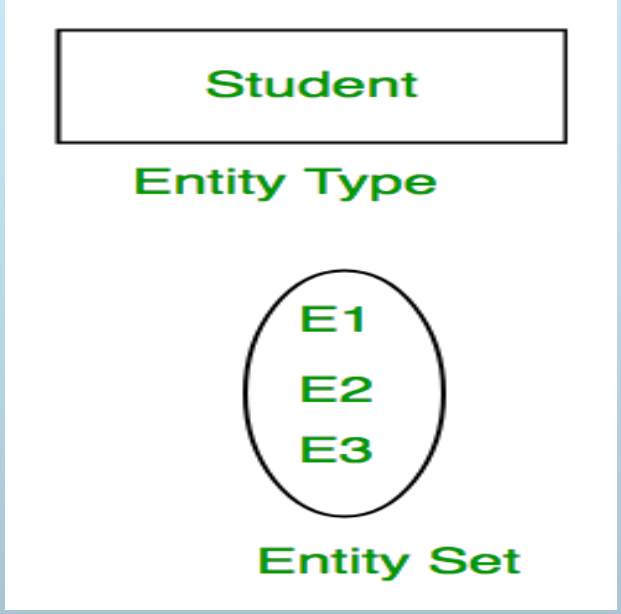
Components of E-R Model



Basic concepts in E-R model

- **1)Entity:**
- An entity can be a real-world object, either animate or inanimate, that can be easily identifiable.

Ex:

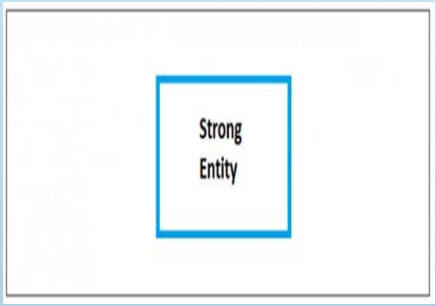


2)Entity set:

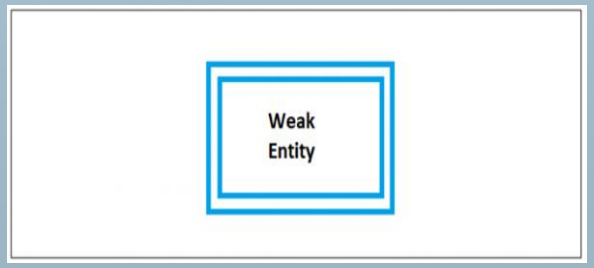
An entity set is a collection of similar types of entities. An entity set may contain entities with attribute sharing similar values.

Basic concepts in E-R model

- **3)Types of Entities:**
- **1) Strong entity:**
- The strong entity has a primary key. Weak entities are dependent on strong entity. Its existence is not dependent on any other entity.



- **2)Weak entity:**
- The weak entity in DBMS do not have a primary key and are dependent on the parent entity. It mainly depends on other entities.
- Weak Entity is represented by double rectangle



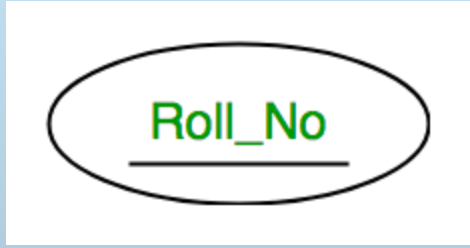
Basic concepts in E-R model

- **4)Attribute:**
- An entity is represented by a set of attributes, that is descriptive properties possessed by all members of an entity set.
- Example:
- *customer = (customer-id, customer-name, customer-street,customer-city) loan = (loan-number, amount)*
- In ER diagram, attribute is represented by an oval.

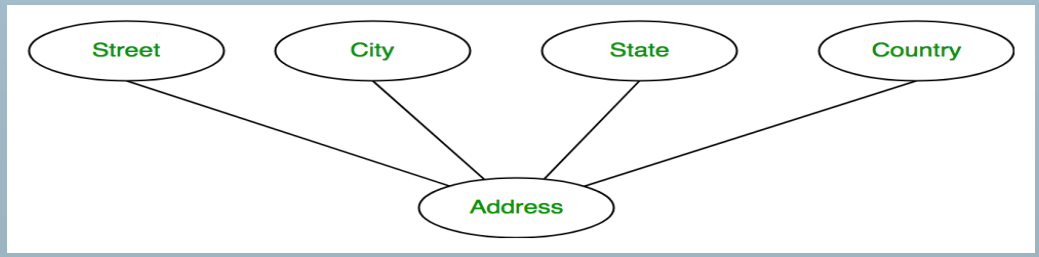
- **Domain** – the set of permitted values for each attribute

Basic concepts in E-R model

- **Types of Attributes:**
- **i) Key Attribute :**
- The attribute which **uniquely identifies each entity** in the entity set is called key attribute.
- In ER diagram, key attribute is represented by an oval with underlying lines.
- **Ex:**



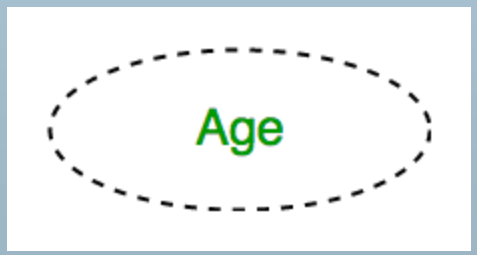
- **ii) Composite Attribute :**
- An attribute **composed of many other attribute** is called as composite attribute.
- In ER diagram, composite attribute is represented by an oval comprising of ovals.



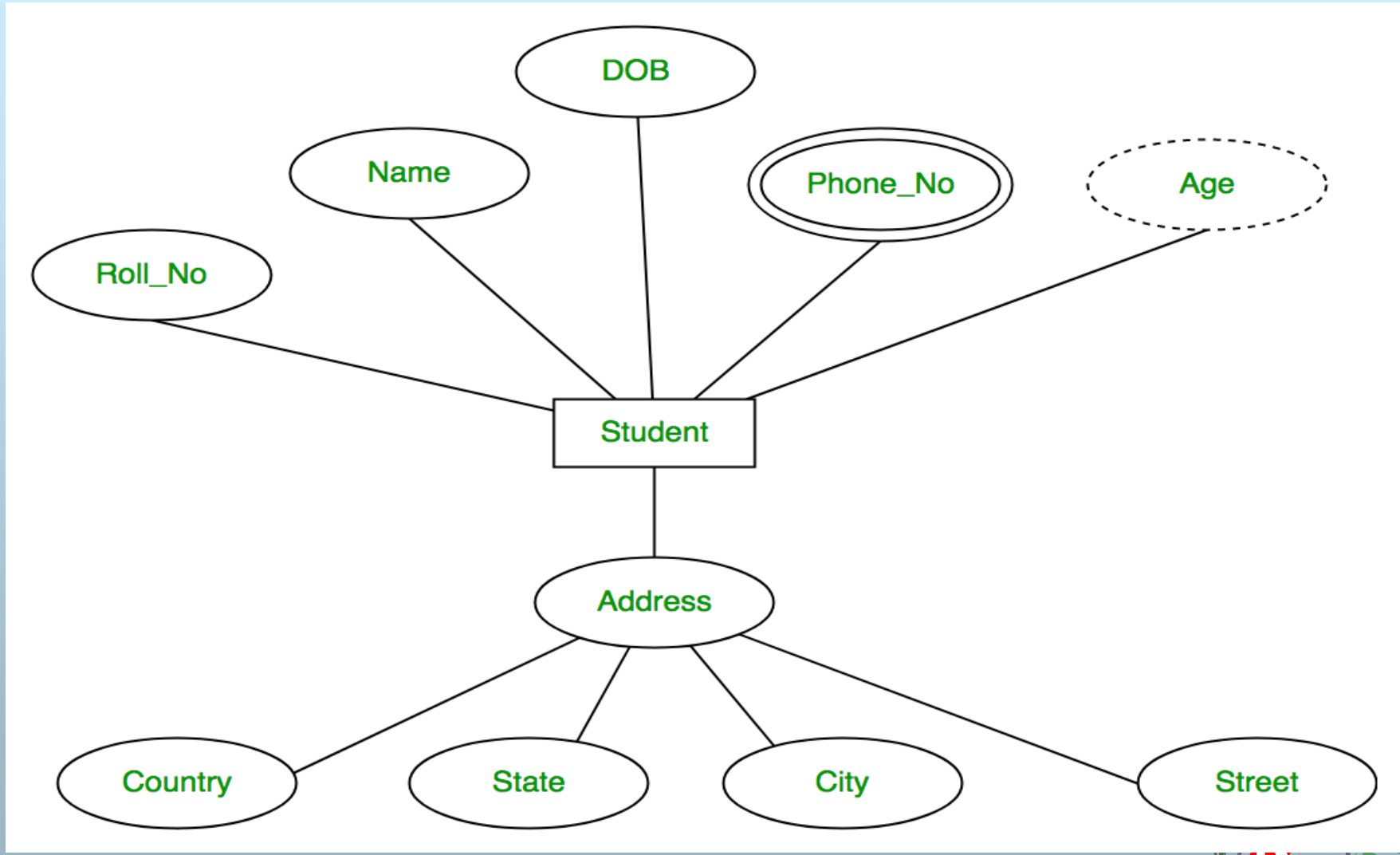
Basic concepts in E-R model

- **iii)Multi valued Attribute :**
- An attribute consisting **more than one value** for a given entity.
- In ER diagram, multivalued attribute is represented by double oval.

- **iv)Derived Attribute :**
- An attribute which can be **derived from other attributes** of the entity type is known as derived attribute.
- In ER diagram, derived attribute is represented by dashed oval.

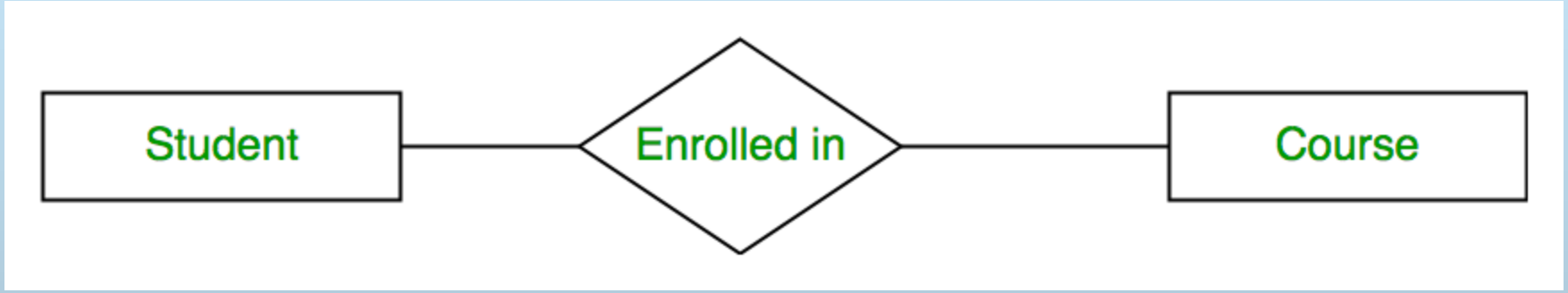


Example



Basic concepts in E-R model

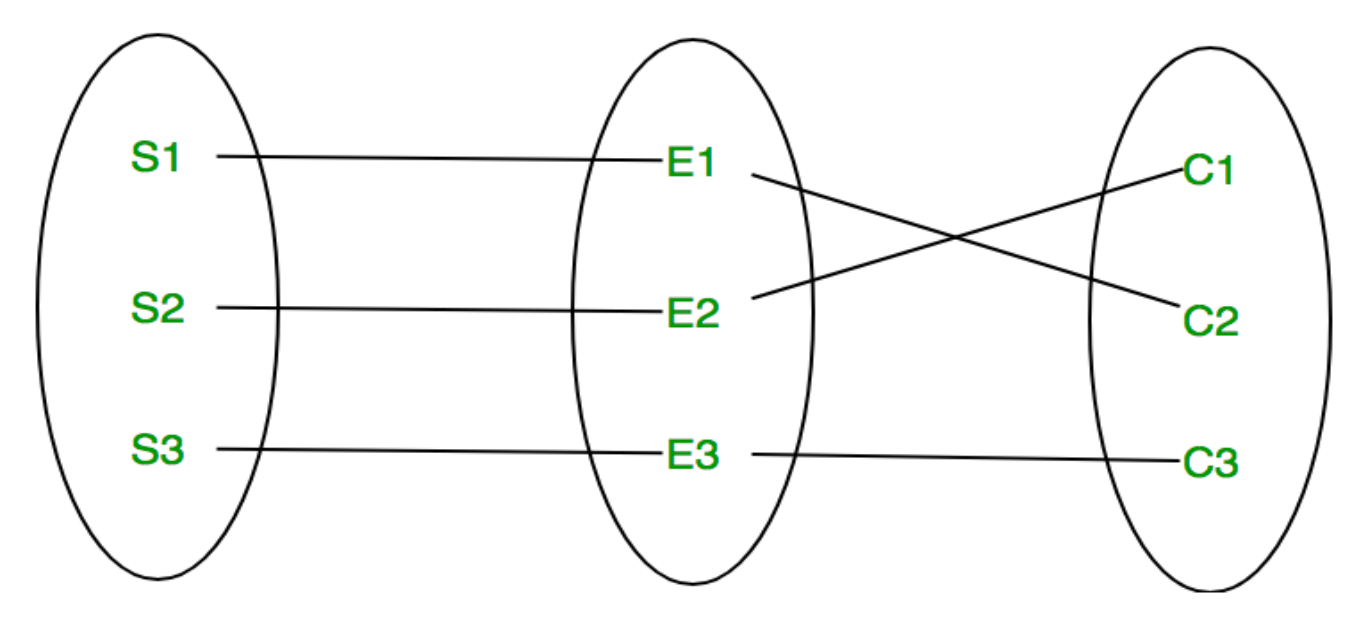
- **5) Relationship Type and Relationship Set:**
- A relationship type represents the **association between entity types**.
- In ER diagram, relationship type is represented by a diamond and connecting the entities with lines.



- Like entities, a relationship too can have attributes. These attributes are called **descriptive attributes**.

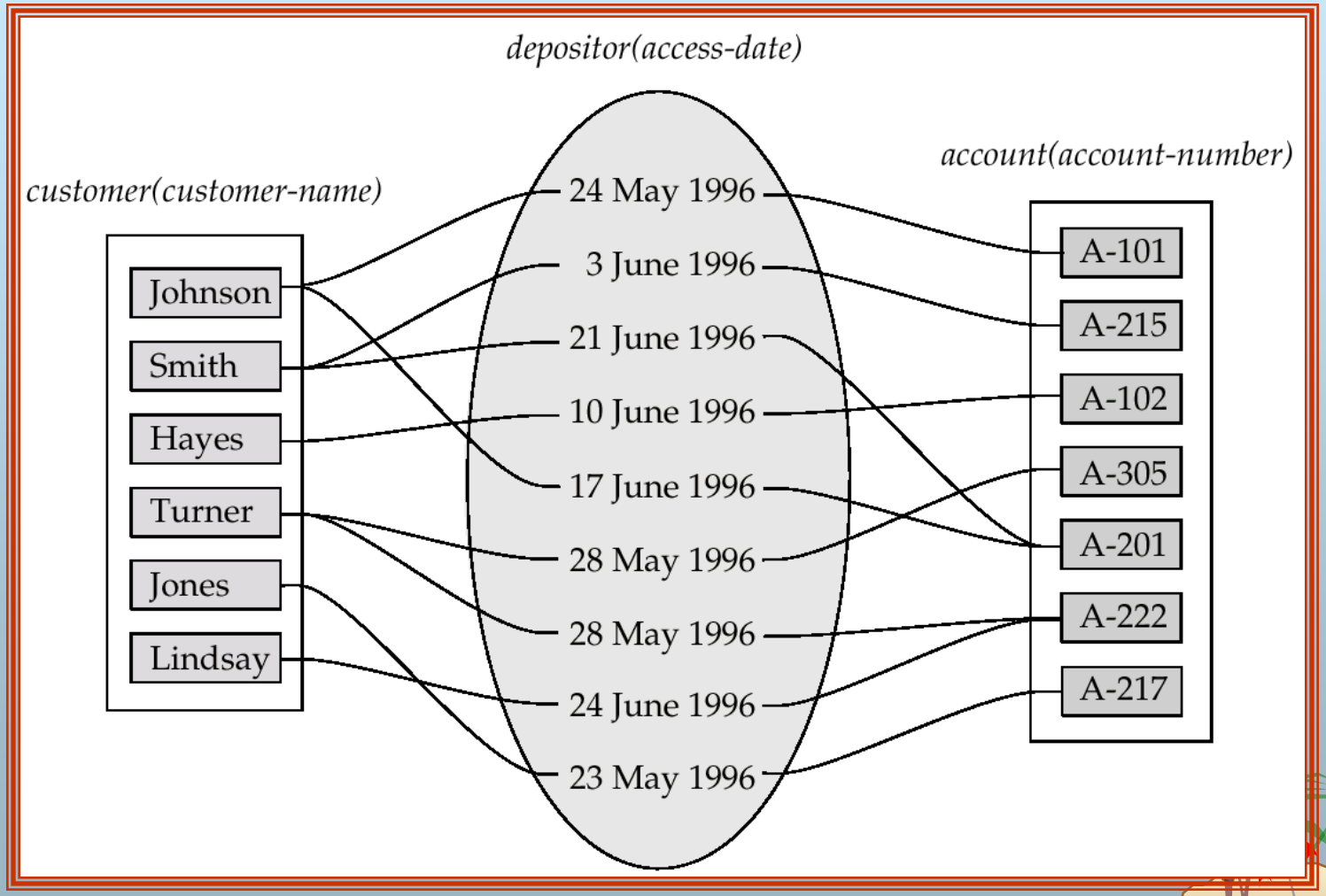
Basic concepts in E-R model

- **Relationship set:**
- A set of relationships of same type is known as relationship set.
- The following relationship set depicts S1 is enrolled in C2, S2 is enrolled in C1 and S3 is enrolled in C3.



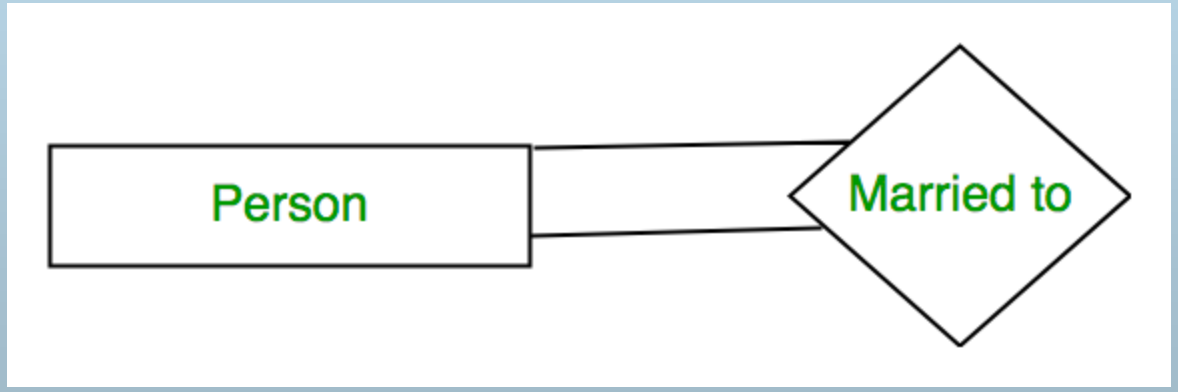
Basic concepts in E-R model

- **Example: Relationship set**



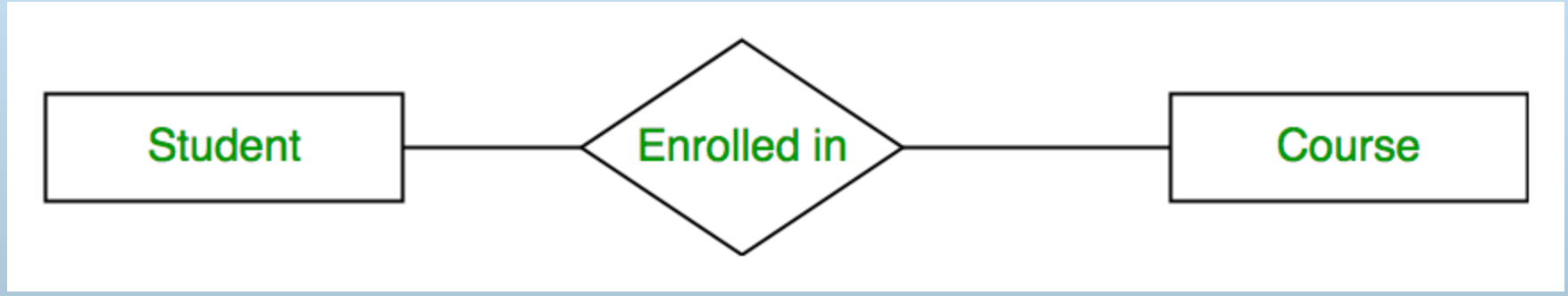
Basic concepts in E-R model

- **Degree of a relationship set:**
  The number of different entity sets **participating in a relationship** set is called as degree of a relationship set.
- **i)Unary Relationship –**
  When there is **only ONE entity set participating in a relation**, the relationship is called as unary relationship.
- For example, one person is married to only one person.



Basic concepts in E-R model

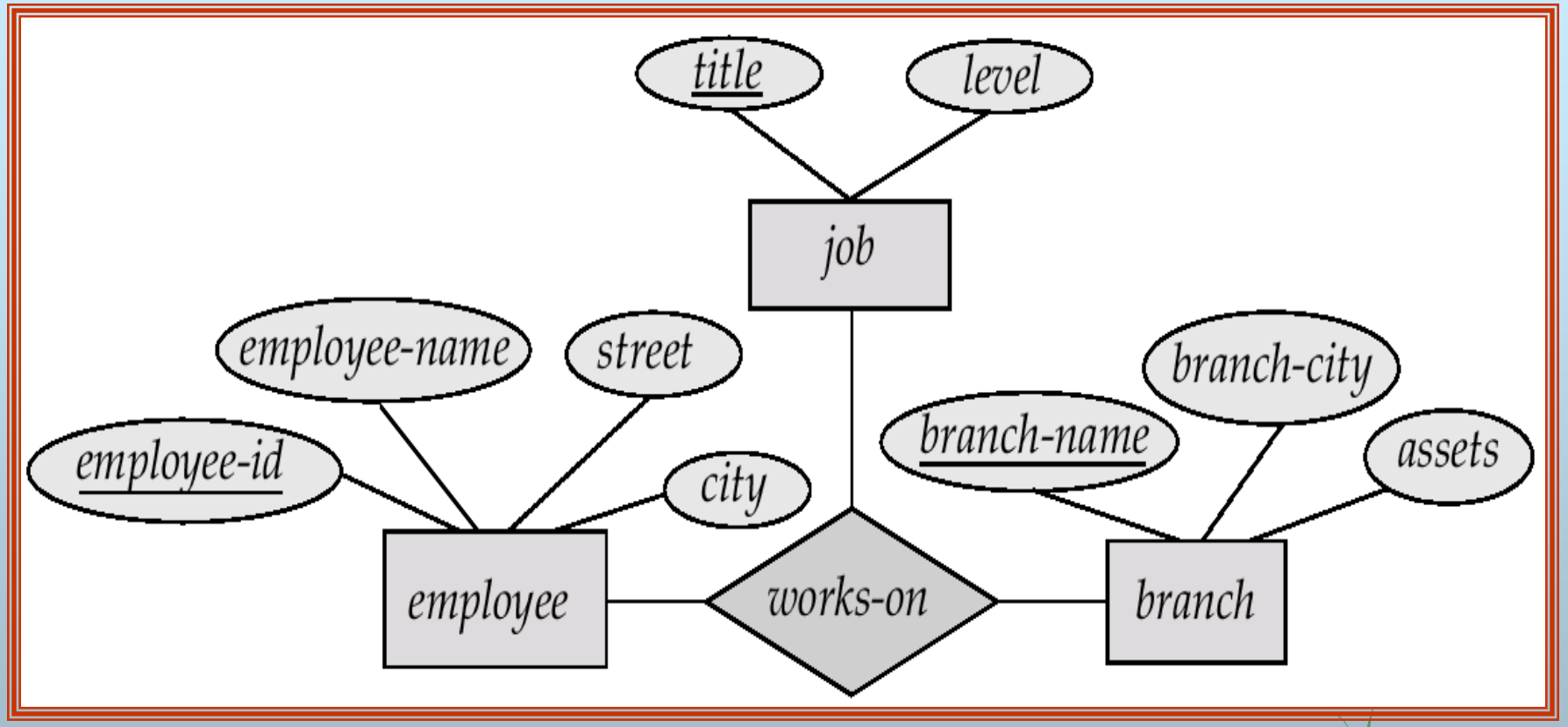
- **ii) Binary Relationship –**
- When there are **TWO entities set participating in a relation**, the relationship is called as binary relationship.
- For example, Student is enrolled in Course.



- **iii)n-ary Relationship –**
- When there are n entities set participating in a relation, the relationship is called as n-ary relationship.

Basic concepts in E-R model

- **Ternary Relationship:**



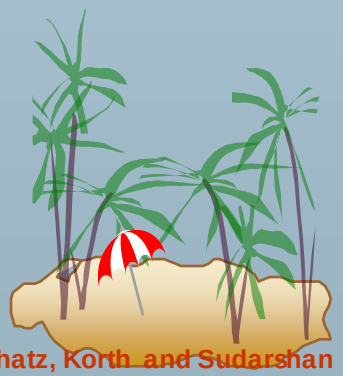
Constraints in ER model

- An E-R enterprise schema may define certain constraints to which the contents of a database must conform.
- Types of constraints are:-

1.Mapping Constraints (Cardinalities)

2.Participation Constraints

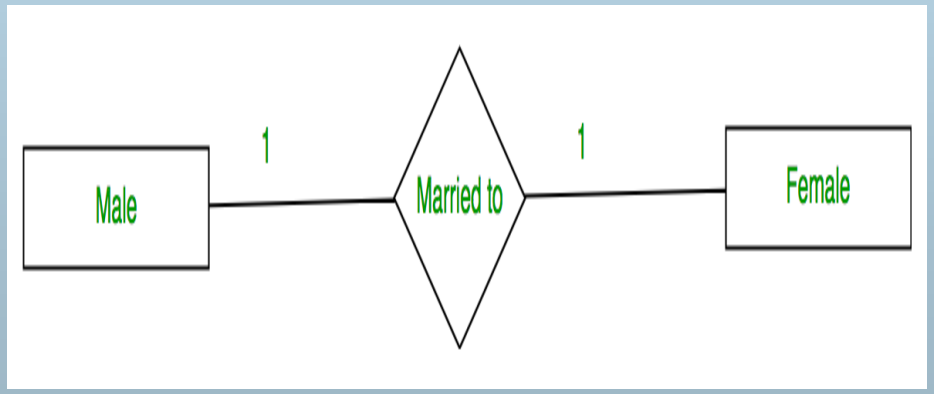
3.Key Constraints

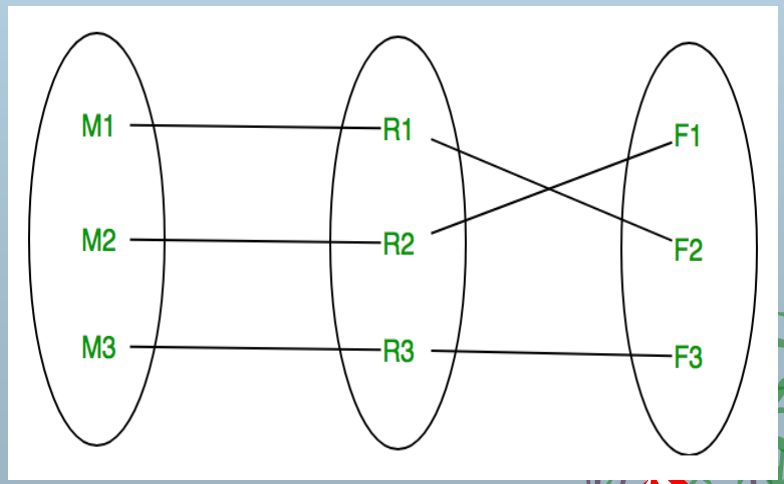


Constraints in ER model

1.Mapping Constraints (Cardinalities)

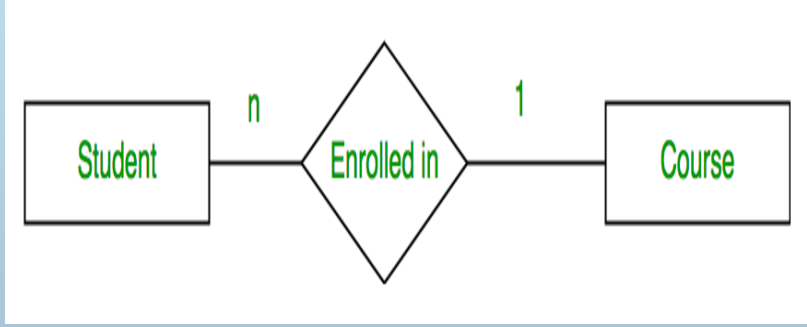
- The **number of times an entity of an entity set participates in a relationship** set is known as cardinality.
- Cardinality can be of different types:
- **i) One to one –**
- When each entity in each entity set can take part **only once in the relationship**, the cardinality is one to one.
- **Ex:**

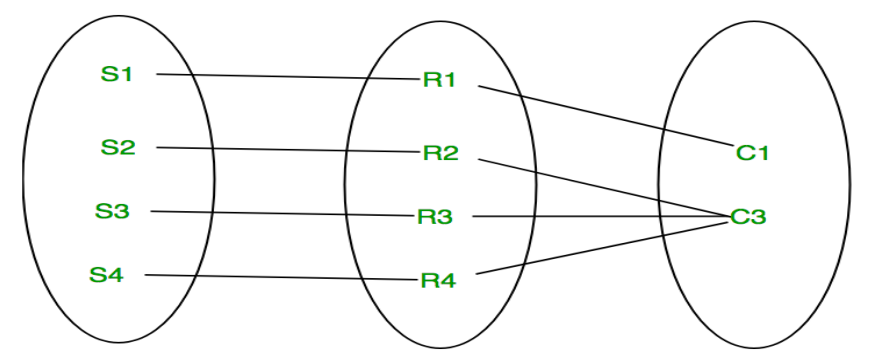




Constraints in ER model

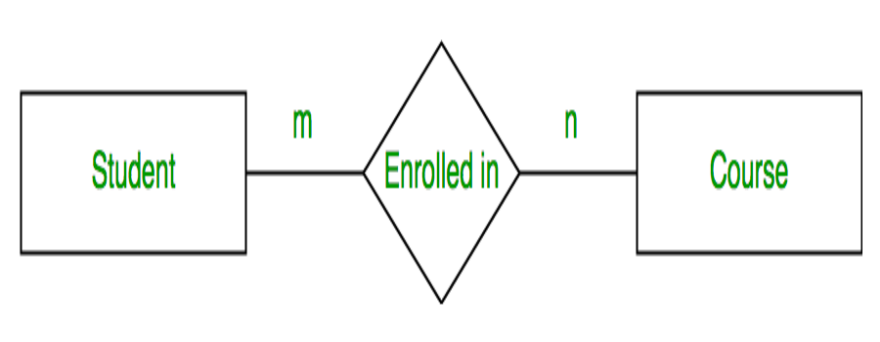
- **li)Many to one –**
- When entities in one entity set **can take part only once in the relationship set and entities in other entity set can take part more than once in the relationship set,** cardinality is many to one.
- **EX:**

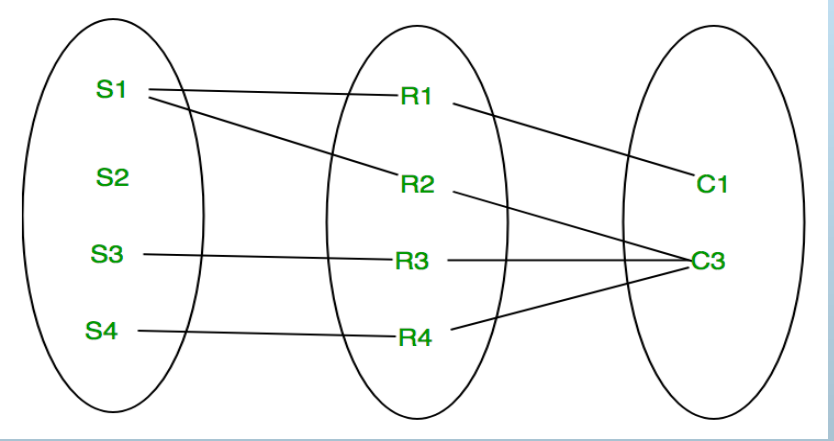




Constraints in ER model

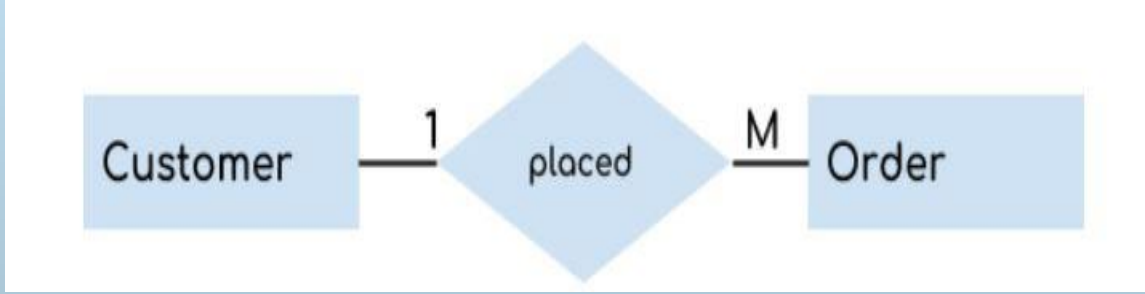
- **iii) Many to many –**
- When entities in all entity sets can **take part more than once in the relationship** cardinality is many to many.
- **Ex:**





Constraints in ER model

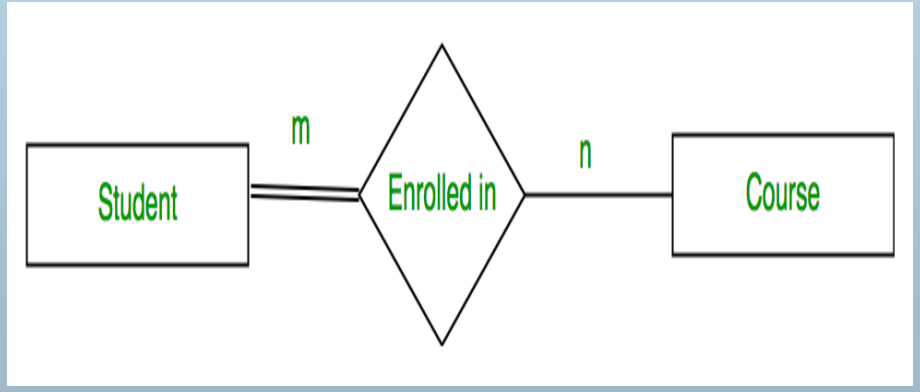
- **Iv)One to Many:**
- When a single instance of an entity is associated with more than one instances of another entity then it is called one to many relationship.
- EX:

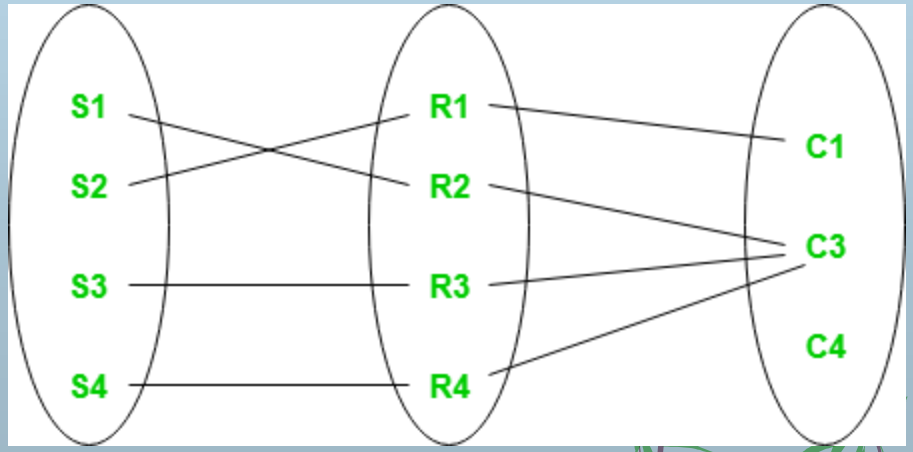


Constraints in ER model

2.Participation Constraints:

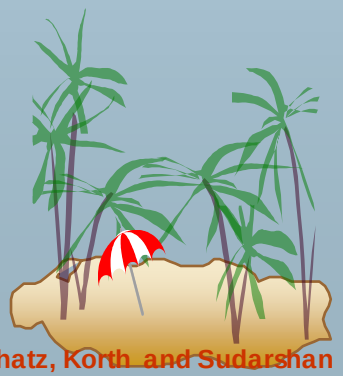
- **Total Participation:** The participation of an entity set *E* in a relationship set *R* is said to be total if every entity in *E* participates in at least one relationship in *R*.
- Total participation is shown by double line in ER diagram.
- **Partial Participation:** If only some entities in *E* participate in relationships in *R*, the participation of entity set *E* in relationship *R* is said to be partial.





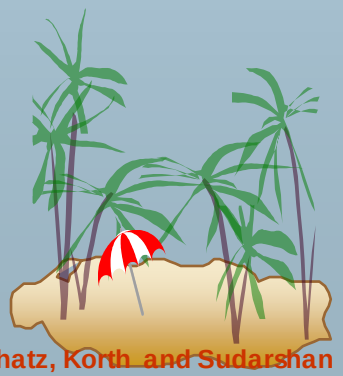
Constraints in ER model

- **3.Key Constraints:**
- Keys are attributes or sets of attributes that uniquely identify an entity within its entity set.
- The values of the attribute values of an entity must be such that they can *uniquely identify* the entity.
- In other words, no two entities in an entity set are allowed to have exactly the same value for all attributes.



Constraints in ER model

- **Types of Key Constraints:**
- Single-value constraints require that a value be unique in certain contexts.
- Referential integrity constrains require that a value referred to actually exists in the database.
- Domain constraints specify what set of values an attribute can take.
- General constraints are arbitrary constraints that should hold in the database.



Constraints in ER model

Keys:

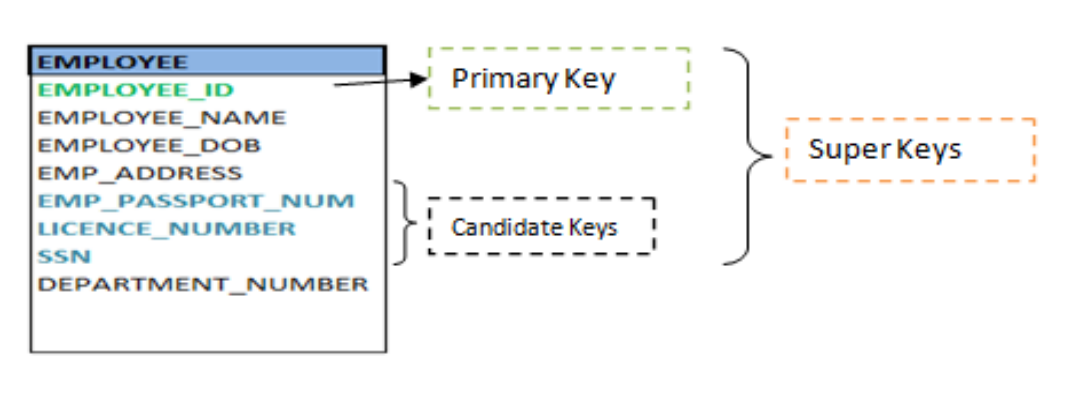
- **Super Key** is the one or more attributes of the entity, which uniquely identifies the record in the database.
- **Candidate Key** is one or more set of keys of the entity.

For a person entity, his SSN, passport#, license# etc can be a candidate key.

- **Primary Key** is the candidate key, which will be used to uniquely identify a record by the query.

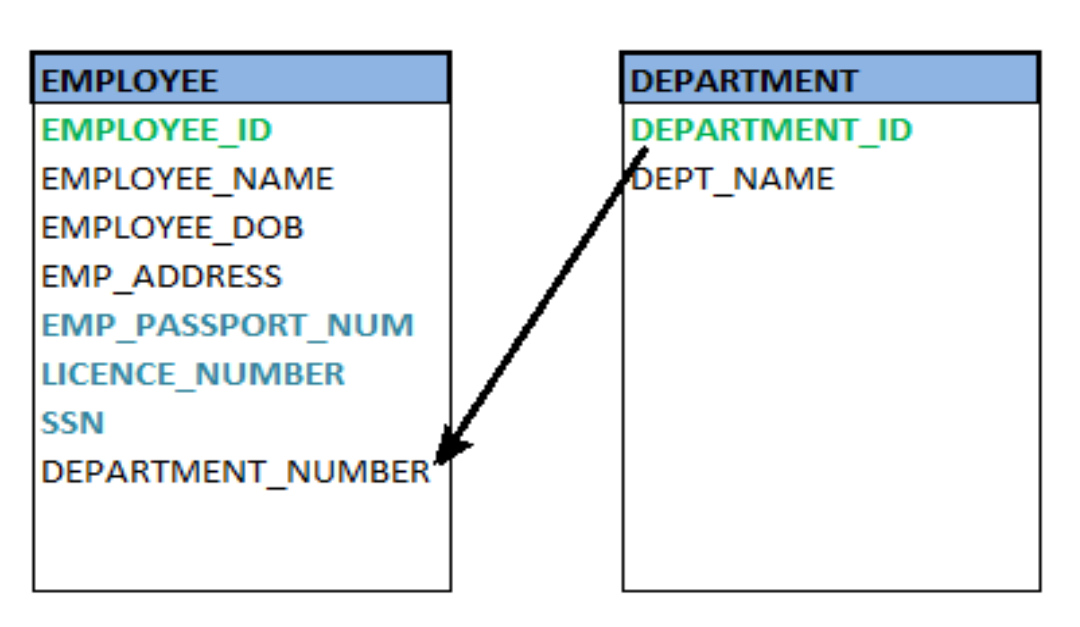
Though a person can be identified using his SSN, passport# or license#, one can choose any one of them as primary key to uniquely identify a person. Rest of them will act as a candidate key.

- **Person:**



Constraints in ER model

- **Referential Integrity constraint or Foreign Key** of the entity attribute in the entity which is the primary key of the related entity. Foreign key helps to establish the mapping between two or more entities.



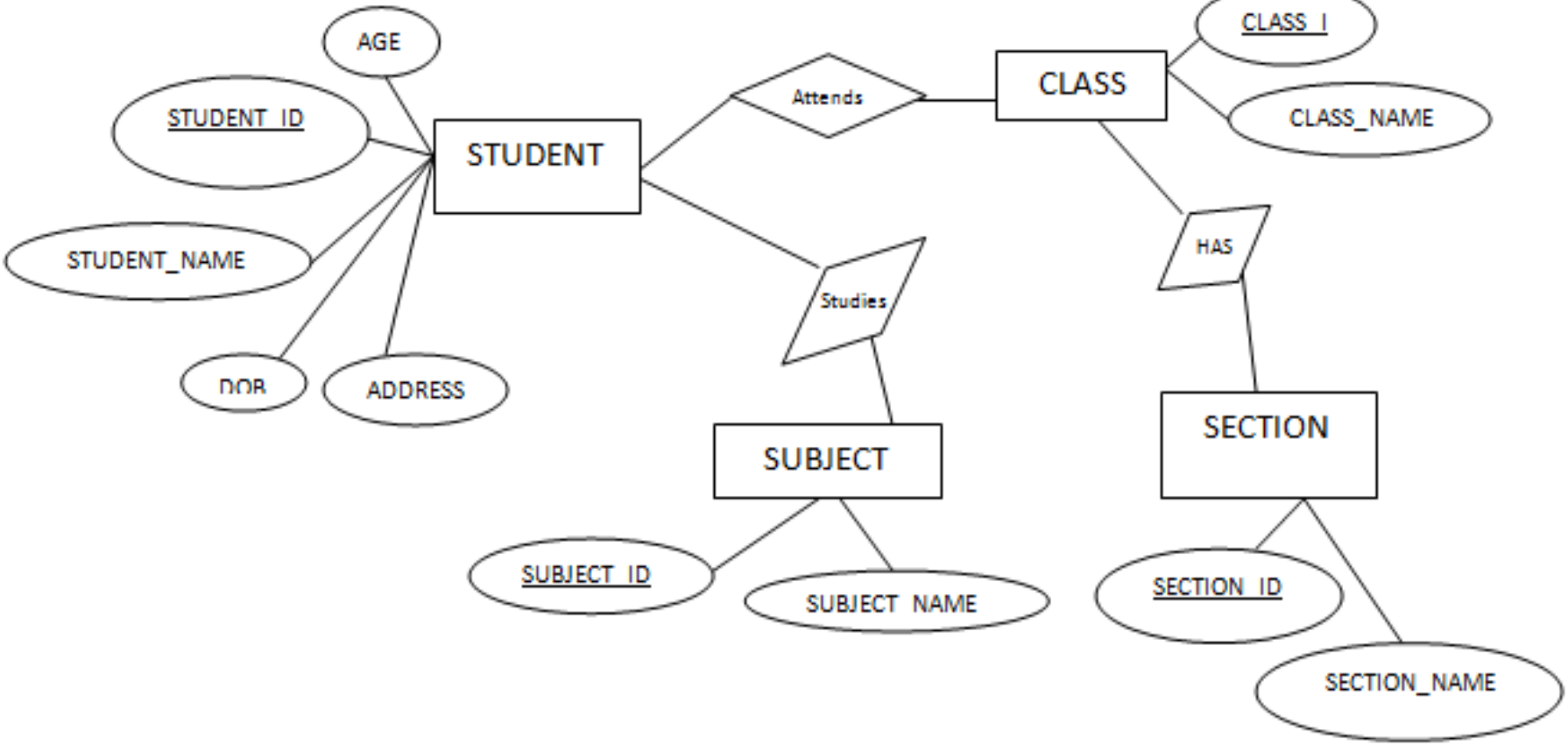
Complete ER diagram : Example

- Example: Construct a simple ER diagram for a STUDENT database
- **Requirement analysis :**
- Student attends class. Each class is divided into one or more sections. Each class will have its own specified subjects. Students have to attend all the subjects of the class that he attends.
- **Conceptual db design:**
- 1)Entities:STUDENT, CLASS, SECTION, SUBJECT
- 2)Attributes:

STUDENT STUDENT_ID STUDENT_NAME ADDRESS DOB AGE CLASS_ID SECTION_ID	CLASS CLASS_ID CLASS_NAME	SECTION SECTION_ID SECTION_NAME	SUBJECT SUBJECT_ID SUBJECT_NAME

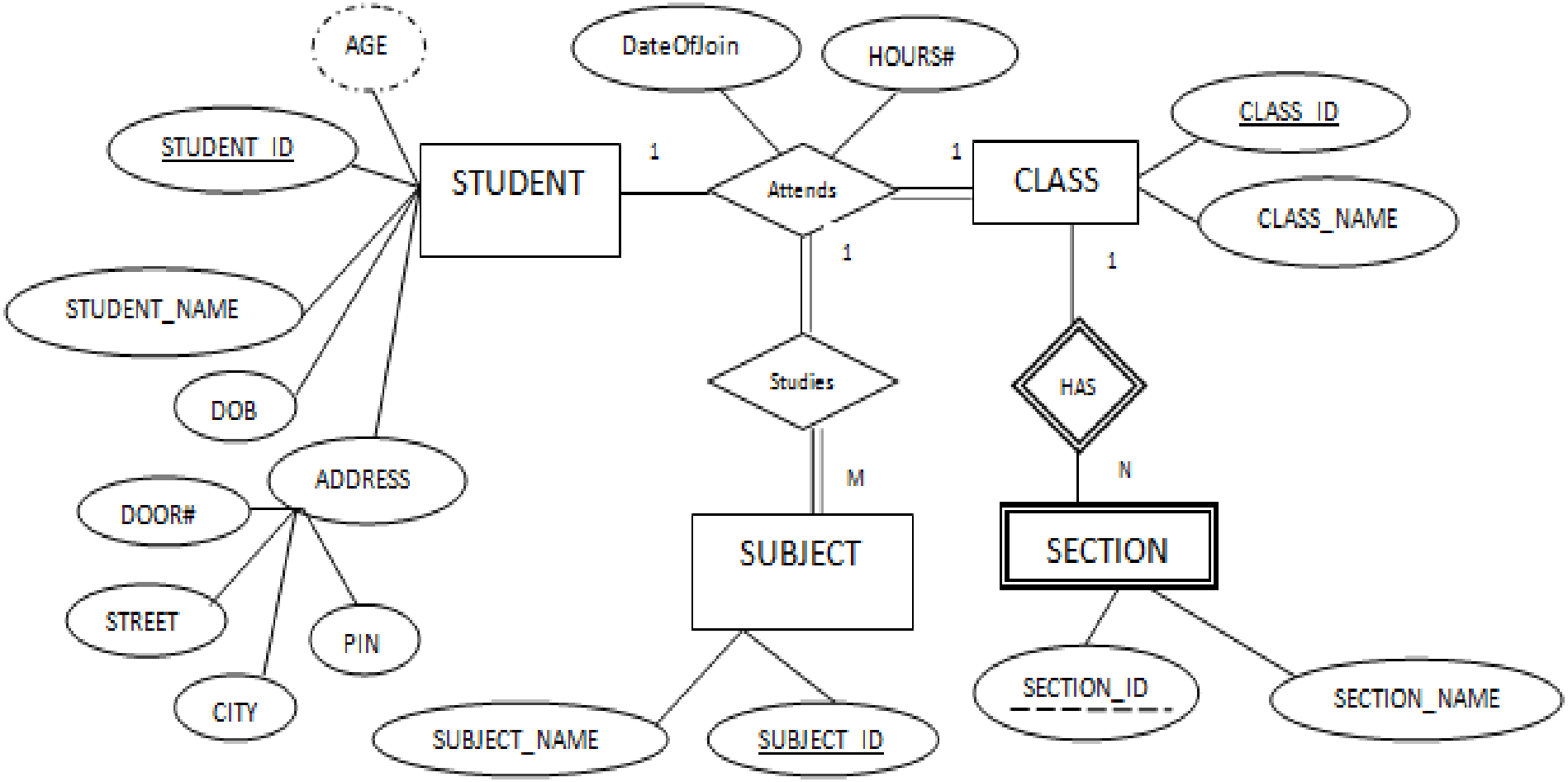
Example

- Relationships:
- ‘Attends’, ‘has section’, ‘have subjects’ and ‘studies subjects’



Complete ER diagram

- After applying all the constraints:



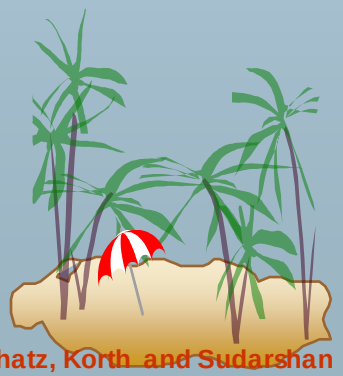
SQL: Integrity constraints

Integrity Constraints:

- Integrity constraints are a set of rules. It is used to maintain the quality of information.
- Integrity constraints ensure that the data insertion, updating, and other processes have to be performed in such a way that data integrity is not affected.
- Constraints can be divided into the following two types,
- **Column level constraints:** Limits only column data.
- **Table level constraints:** Limits whole table data.

Types of Integrity Constraints:

- **Domain Integrity constraints**
- **Entity Integrity constraints**
- **Referential Integrity Constraints**



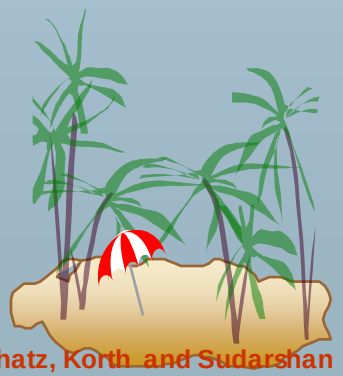
Domain Constraints

- **Domain constraints can be defined as the definition of a valid set of values for an attribute.**
- The data type of domain includes string, character, integer, time, date, currency, etc. The value of the attribute must be available in the corresponding domain.
- – Field: Not Null, Check, Unique, Primary Key, Foreign Key

Example:

ID	NAME	SEMENSTER	AGE
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1004	Morgan	8th	A

Not allowed. Because AGE is an integer attribute



Entity Integrity constraints

- **The entity integrity constraint states that primary key value can't be null and it should contain unique values.**

The primary key value is used to identify individual rows in relation and if the primary key has a null value, then we can't identify those rows.

- **Table: Check, Unique, Primary Key, Foreign Key**
- **Example:**

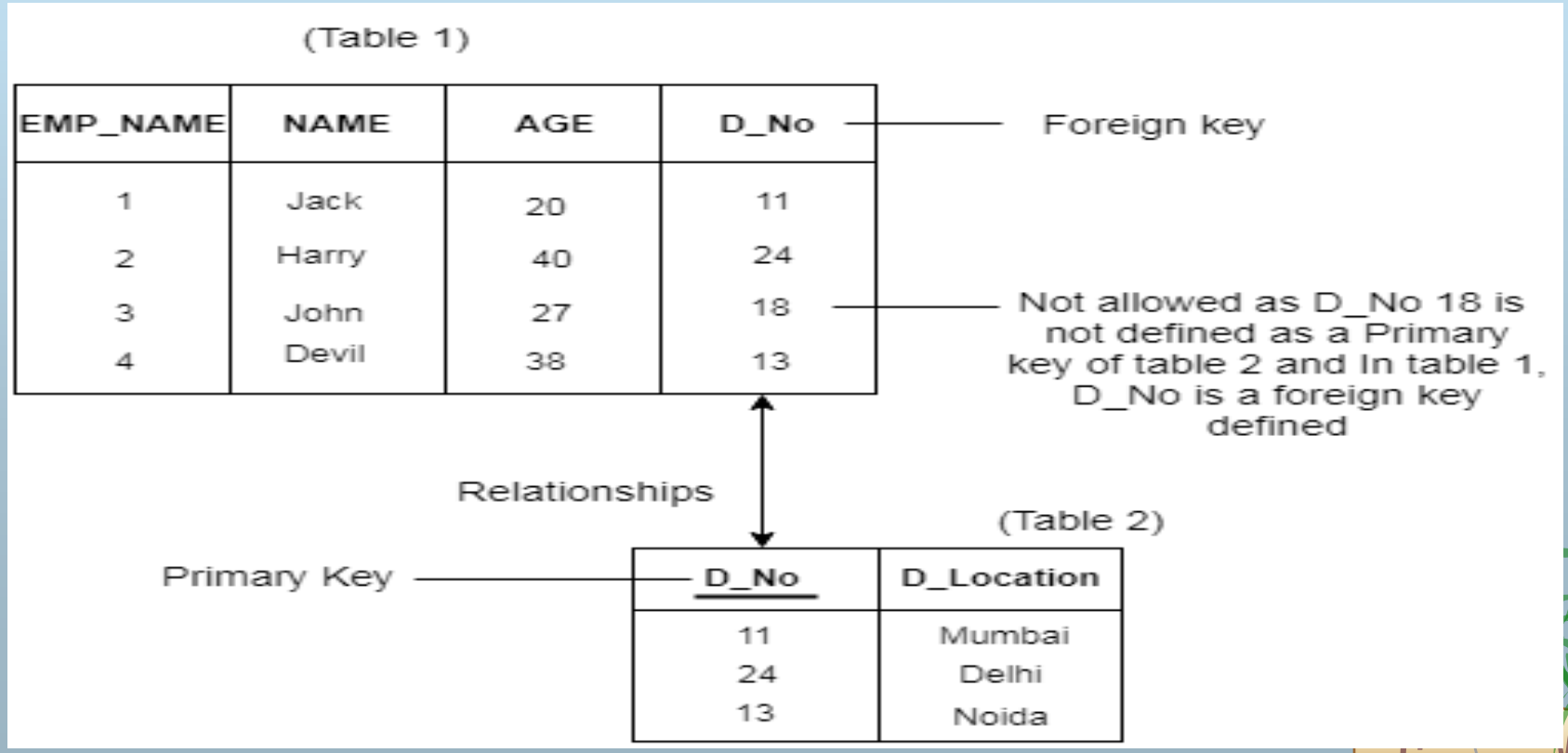
EMPLOYEE

EMP_ID	EMP_NAME	SALARY
123	Jack	30000
142	Harry	60000
164	John	20000
	Jackson	27000

Not allowed as primary key can't contain a NULL value

Referential Integrity Constraints

- A referential integrity constraint is specified between two tables.
- In the Referential integrity constraints, if a foreign key in Table 1 refers to the Primary Key of Table 2, then every value of the Foreign Key in Table 1 must be null or be available in Table 2.
- Example:



Specification of constraints in SQL

- 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
- **Syntax: (Column level)**

```
CREATE TABLE sample_table (
column1 data_type(size) constraint_name,
column2 data_type(size) constraint_name,
column3 data_type(size) constraint_name,
.... );
```

Specification of constraints in SQL

- Syntax: (Table level)
- CREATE TABLE sample_table (

  column1 data_type(size),

  column2 data_type(size),

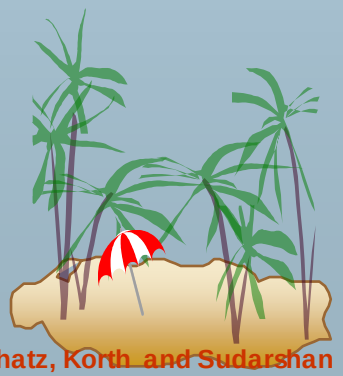
  column3 data_type(size), ...

  CONSTRAINT userdefined_constraint_name UNIQUE/ PRIMARY KEY/ CHECK (column1,COLUMN2…)

);

SQL: NOT NULL

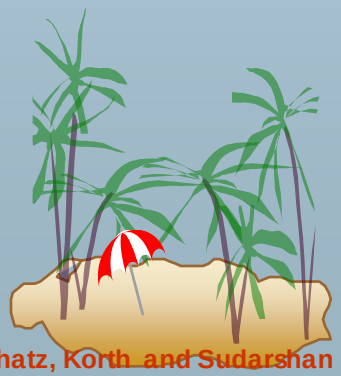
- **NOT NULL Constraint:**
- The NOT NULL constraint enforces a column to NOT accept NULL values.
- This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.
- **Example:(Column level constraint)**
-
```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255) NOT NULL,
    Age int
);
```
- **NOT NULL on ALTER TABLE:**
- ALTER TABLE Persons MODIFY Age int NOT NULL;



SQL: UNIQUE

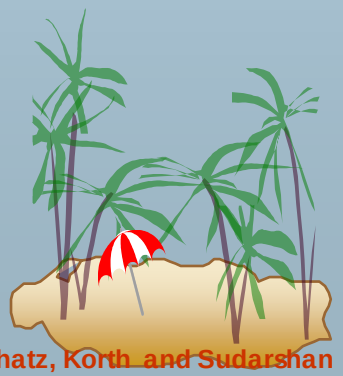
- **UNIQUE Constraint:**
- The UNIQUE constraint ensures that all values in a column are different.
- Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns.
- A PRIMARY KEY constraint automatically has a UNIQUE constraint.
- However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.
- **UNIQUE Constraint on CREATE TABLE(Column level Constraint)**
- **Example:**

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



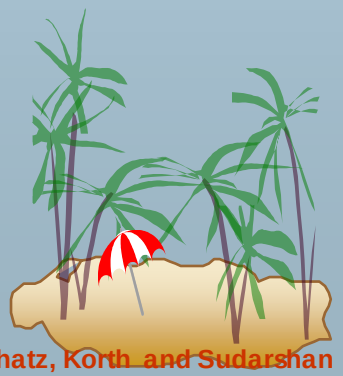
SQL: UNIQUE

- UNIQUE constraint on multiple columns(Table Level constraint)
- **Example:**
- CREATE TABLE Persons (
  ID int NOT NULL,
  LastName varchar(255) NOT NULL,
  FirstName varchar(255),
  Age int,
  CONSTRAINT UC_Person UNIQUE (ID,LastName)
  );



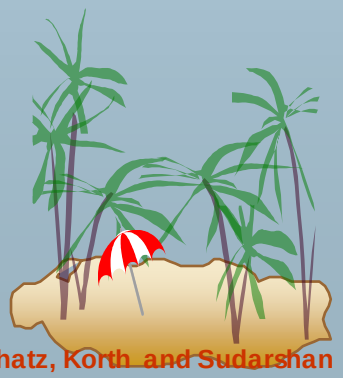
SQL: UNIQUE

- **UNIQUE Constraint on Alter Table:**
- **Column level:**
- ALTER TABLE Persons ADD UNIQUE (ID);
- **Table level:**
- ALTER TABLE Persons
  ADD CONSTRAINT UC_Person UNIQUE (ID,LastName);
- **DROP a UNIQUE Constraint**
- ALTER TABLE Persons DROP CONSTRAINT UC_Person;



SQL: PRIMARY KEY Constraint

- The PRIMARY KEY constraint uniquely identifies each record in a table.
- Primary keys must contain UNIQUE values, and cannot contain NULL values.
- A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).
- **SQL PRIMARY KEY on CREATE TABLE:(column level)**
- CREATE TABLE Persons (
  ID int NOT NULL PRIMARY KEY,
  LastName varchar(255) NOT NULL,
  FirstName varchar(255),
  Age int
  );

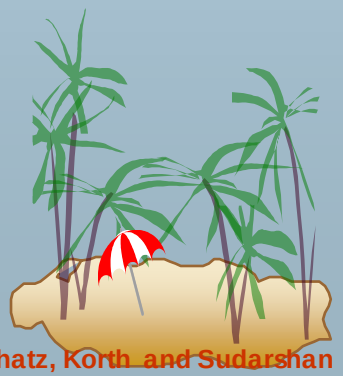


SQL: PRIMARY KEY Constraint

- **SQL PRIMARY KEY on CREATE TABLE:(Table level)**
- CREATE TABLE Persons (
  ID int NOT NULL,
  LastName varchar(255) NOT NULL,
  FirstName varchar(255),
  Age int,
  CONSTRAINT PK_Person PRIMARY KEY (ID,LastName)
  );
- **SQL PRIMARY KEY on ALTER TABLE: (Column level)**
- ALTER TABLE Persons ADD PRIMARY KEY (ID);
- **SQL PRIMARY KEY on ALTER TABLE: (Table level)**
- ALTER TABLE Persons
  ADD CONSTRAINT PK_Person PRIMARY KEY (ID,LastName);

SQL: PRIMARY KEY Constraint

- **DROP a PRIMARY KEY Constraint:**
- ALTER TABLE Persons DROP CONSTRAINT PK_Person;



SQL: 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.
- Look at the following two tables:
- Persons table and Orders table:

PersonID	LastName	FirstName	Age
1	Hansen	Ola	30
2	Svendson	Tove	23
3	Pettersen	Kari	20

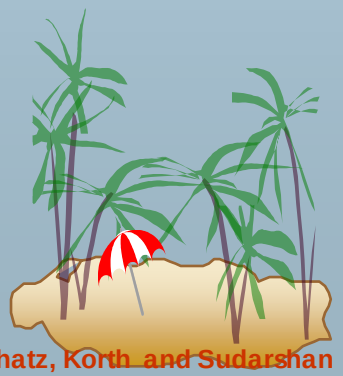
OrderID	OrderNumber	PersonID
1	77895	3
2	44678	3
3	22456	2
4	24562	1

SQL: FOREIGN KEY Constraint

- **SQL FOREIGN KEY on CREATE TABLE: (Column level)**
- CREATE TABLE Orders (
  OrderID int NOT NULL PRIMARY KEY,
  OrderNumber int NOT NULL,
  PersonID int FOREIGN KEY REFERENCES Persons(PersonID)
  );
- **SQL FOREIGN KEY on CREATE TABLE: (Table level)**
- CREATE TABLE Orders (
  OrderID int NOT NULL,
  OrderNumber int NOT NULL,
  PersonID int,
  PRIMARY KEY (OrderID),
  CONSTRAINT FK_PersonOrder FOREIGN KEY (PersonID)
  REFERENCES Persons(PersonID)
  );

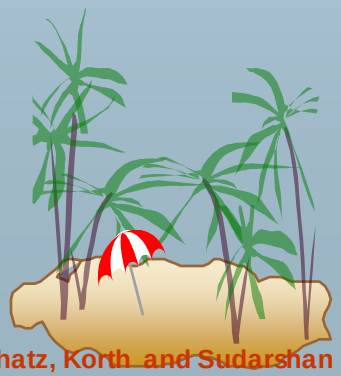
- **SQL FOREIGN KEY on ALTER TABLE: (Column level)**
- ALTER TABLE Orders
  ADD FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);
- **SQL FOREIGN KEY on ALTER TABLE: (Table level)**
- ALTER TABLE Orders
  ADD CONSTRAINT FK_PersonOrder
  FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);

- **DROP a FOREIGN KEY Constraint:**
- ALTER TABLE Orders
  DROP CONSTRAINT FK_PersonOrder;



SQL : CHECK Constraint

- The CHECK constraint is used to limit the value range that can be placed in a column.
- If you define a CHECK constraint on a single column it allows only certain values for this column.
- If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.
- **SQL CHECK on CREATE TABLE: (column level)**
- CREATE TABLE Persons (
  ID int NOT NULL,
  LastName varchar(255) NOT NULL,
  FirstName varchar(255),
  Age int CHECK (Age>=18)
  );

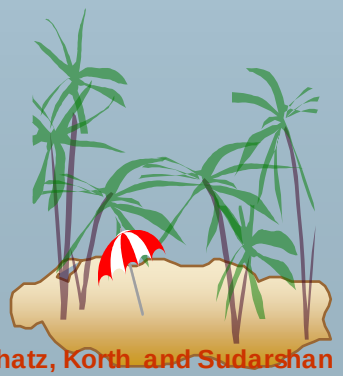


SQL : CHECK Constraint

- **SQL CHECK on CREATE TABLE: (Table level)**
- CREATE TABLE Persons (
  ID int NOT NULL,
  LastName varchar(255) NOT NULL,
  FirstName varchar(255),
  Age int,
  City varchar(255),
  CONSTRAINT CHK_Person CHECK (Age>=18 AND City='hyd'));
- **SQL CHECK on ALTER TABLE: (column level)**
- ALTER TABLE Persons ADD CHECK (Age>=18);
- **SQL CHECK on ALTER TABLE: (Table level)**
- ALTER TABLE Persons
  ADD CONSTRAINT CHK_PersonAge CHECK (Age>=18 AND City='hyd'
  );
- **DROP a CHECK Constraint:**
- ALTER TABLE Persons DROP CONSTRAINT CHK_PersonAge;

SQL: DEFAULT Constraint

- The DEFAULT constraint is used to provide a default value for a column.
- The default value will be added to all new records IF no other value is specified.
- **SQL DEFAULT on CREATE TABLE: (Column level)**
- CREATE TABLE Persons (
  ID int NOT NULL,
  LastName varchar(255) NOT NULL,
  FirstName varchar(255),
  Age int,
  City varchar(255) DEFAULT ‘hyderabad'
  );



SQL: DEFAULT Constraint

- **SQL DEFAULT on ALTER TABLE:**
- ALTER TABLE Persons
  MODIFY City DEFAULT ‘hyderabad';
- **DROP a DEFAULT Constraint:**
- ALTER TABLE Persons
  ALTER COLUMN City DROP DEFAULT;