# DATABASE MANAGEMENT SYSTEMS

**Course code:ACSB08**

**B.Tech IV semester**

**Regulation: IARE R-18**


BY

Mr. U. Sivaji

Assistant Professors

Mr. N PoornaChandra Rao, Mr. N Bhaswanth, Ms. B Ramya sree, Ms. K Mayuri, Ms. B Vijaya Durga

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**INSTITUTE OF AERONAUTICAL ENGINEERING**

**(Autonomous)**

**DUNDIGAL, HYDERABAD - 500 043**

| CO's | Course outcomes |
|---|---|
| CO1 | Describe purpose of database systems, languages, users, architecture and models such as ER model, relational model. |
| CO2 | Determine relational algebra, relational calculus and expressive power of algebra and calculus. |
| CO3 | Understand sql – data definition commands, manipulation commands, relational database design and normal forms. |
| CO4 | Explore the concept of Transaction States, Concurrency Control, Deadlock Handling and Recovery mechanisms. |
| CO5 | Knowledge about the Physical Storage Media, Indexing and Hashing, and Query Processing. |


# MODULE– I
# CONCEPTUAL MODELING INTRODUCTION LAPLACE

| CLOs | Course Learning Outcome |
|---|---|
| CLO1 | Describe the Purpose of Database Systems, Data Models, and View of Data. |
| CLO2 | Summarize the concept of Database Languages, Database Users. |
| CLO3 | Identify the Various Components of overall DBS architecture. |
| CLO4 | Use the concept of ER Model. |
| CLO5 | Describe Basics of Relational Model. |

# INTRODUCTION TO DATA BASES

- DBMS contains information about a particular enterprise
  - Collection of interrelated data
  - Set of programs to access the data
  - 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
  - Online retailers: order tracking, customized recommendations
  - Manufacturing: production, inventory, orders, supply chain
  - Human resources: employee records, salaries, tax deductions

# PURPOSE OF DATABASE SYSTEMS

- In the early days, database applications were built directly on top of file systems
- Drawbacks of using file systems to store data:
    - Data redundancy and inconsistency
    - Difficulty in accessing data
    - Data isolation — multiple files and formats
    - Integrity problems
    - Atomicity of updates
        - Example: Transfer of funds from one account to another should either complete or not happen at all
    - Concurrent access by multiple users
        - Example: Two people reading a balance and updating it at the same time
    - Security problems

# PURPOSE OF DATABASE SYSTEMS

| | DISADVANTAGES OF FILE SYSTEMS | ADVANTAGES OF DBMS |
|---|---|---|
| 1 | Data v/s program problem:<br>Different programs access different files | One set of programs access all data |
| 2 | Data inconsistency problem<br>As same data resides in many different files across the programs data inconsistency increases | Related data resides in same storage location minimizing data inconsistency |
| 3 | Data isolation problem<br>As data is scattered in various files and in different formats it is difficult to write new programs to retrieve appropriate data | As data resides in same storage location it is easy to write new programs to retrieve appropriate data |
| 4 | Security problem:Every user can acces all data | Every user can access only needed data |

# PURPOSE OF DATABASE SYSTEMS

| | | |
|---|---|---|
| 5 | Integrity problem:Develop new consistent range in exixting systems appropriate code must be added in various application program | Integrity Solution:appropriate code must be added in one application program that access all data at one time |
| 6 | Problem in accessing data:new appropriate program has to be written each time | DBMS consists of one or more programs to extract needed information |
| 7 | Atomicity problem:If system fails it must ensure data are restored to consistent state | It ensures atomicity |
| 8 | Data Redundancy:same information is dupliacated in several files ,so higher storage and access cost | One copy of data resides so minimium storage and access cost |
| 9 | Concurrency problem:Due to redundant data if many users access same copy leads to concurrency problem | Avoids concurrency problem since data last changed remains permanent |

# VIEW OF DATA

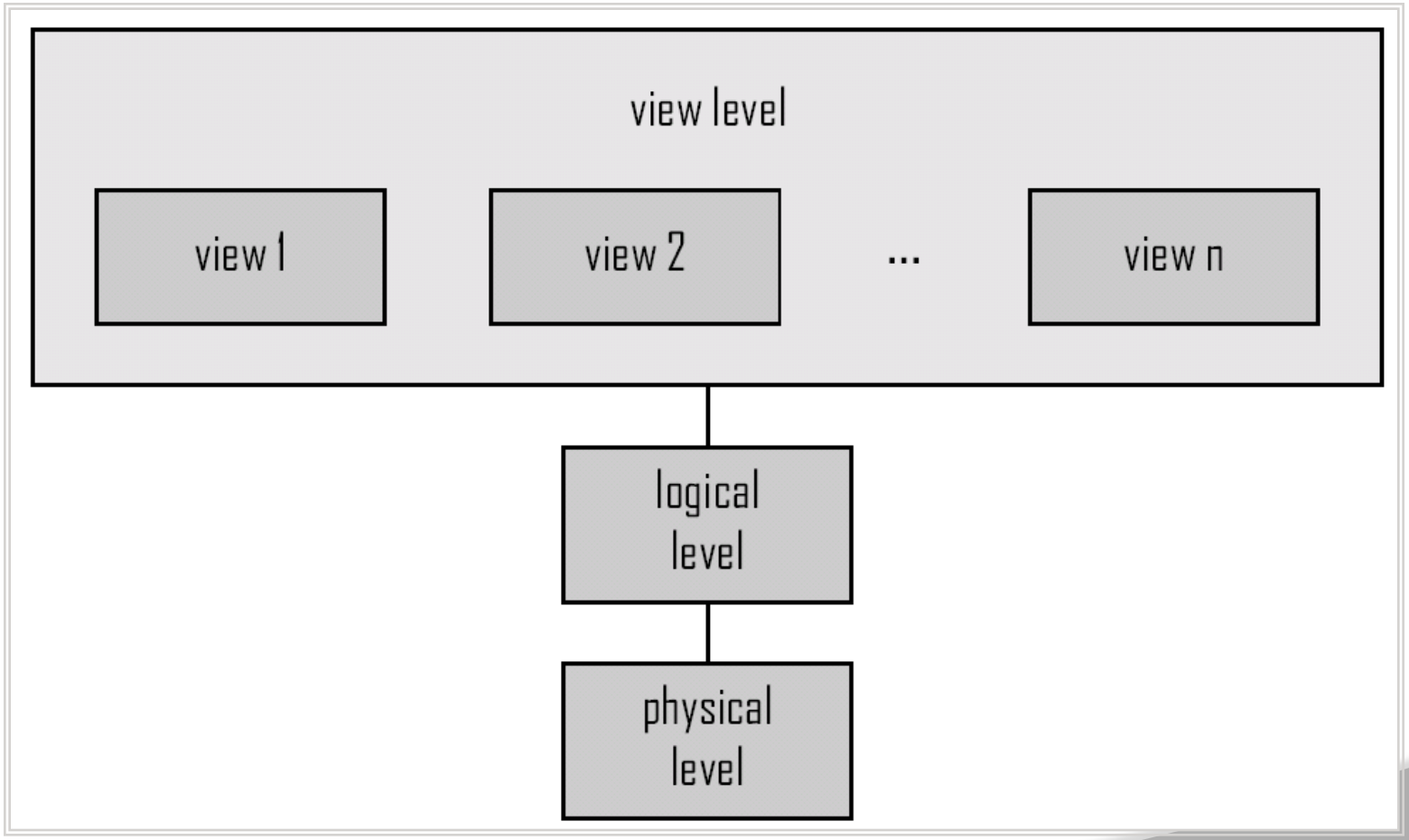
- Physical level: describes how a record (e.g., customer) is stored.
- Logical level: describes data stored in database, and the relationships among the data.

```
type customer = record
        customer_id:string;
        customer_name:string;
        customer_street:string;
        customer_city : string;
end;
```

- View level: application programs hide details of data types. Views can also hide information (such as an employee's salary) for security purposes.

# VIEW OF DATA

An architecture for a database system



# DATA MODELS

Underlying the structure of database is data model.

It is a collection of tools for describing

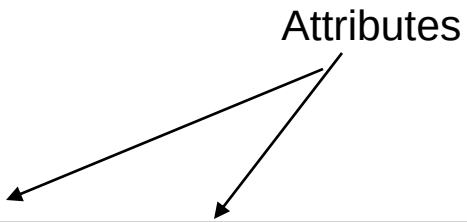
- Data ,Data relationships, Data semantics & consistency constraints
- Data model types

Relational model

- Entity-Relationship data model (mainly for database design)
- Object-based data models (Object-oriented and Object-relational)
- Semi structured data model (XML)
- Other older models:
- Network model
- Hierarchical model

# DATA MODELS

- Example of tabular data in the relational model




| *customer_id* | *customer_name* | *customer_street* | *customer_city* | *account_number* |
|---|---|---|---|---|
| 192-83-7465 | Johnson | 12 Alma St. | Palo Alto | A-101 |
| 192-83-7465 | Johnson | 12 Alma St. | Palo Alto | A-201 |
| 677-89-9011 | Hayes | 3 Main St. | Harrison | A-102 |
| 182-73-6091 | Turner | 123 Putnam St. | Stamford | A-305 |
| 321-12-3123 | Jones | 100 Main St. | Harrison | A-217 |
| 336-66-9999 | Lindsay | 175 Park Ave. | Pittsfield | A-222 |
| 019-28-3746 | Smith | 72 North St. | Rye | A-201 |

# DATA MODELS

- A Sample Relational Database

| *customer_id* | *customer_name* | *customer_street* | *customer_city* |
|---|---|---|---|
| 192-83-7465 | Johnson | 12 Alma St. | Palo Alto |
| 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 |

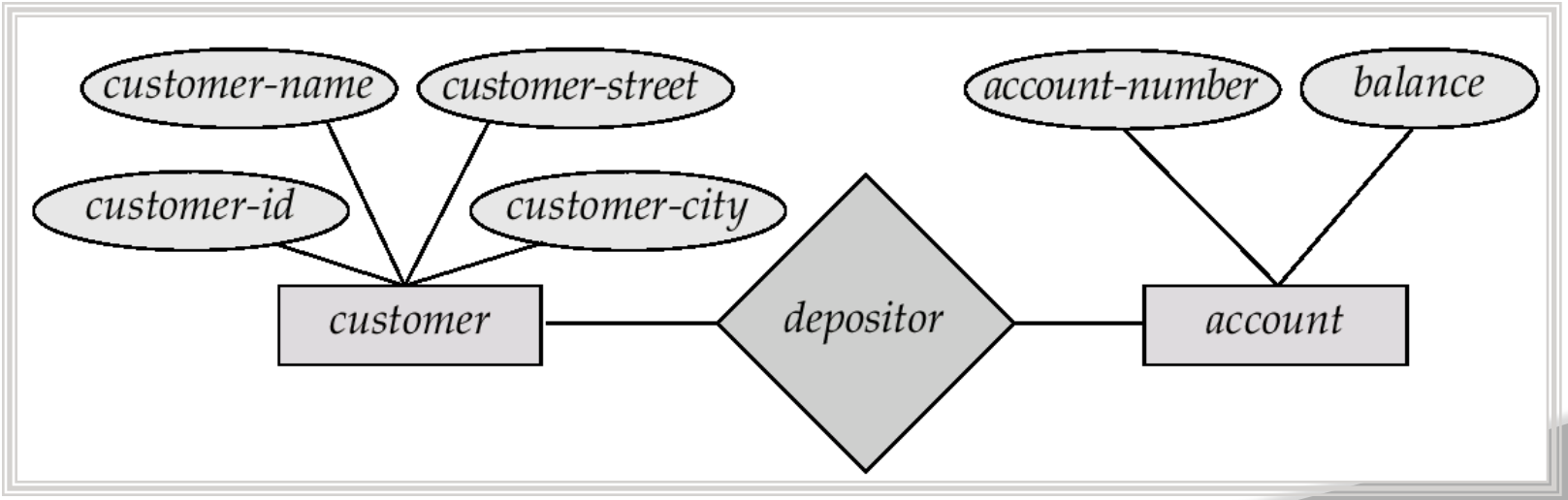
(b) The *account* table

| *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

# DATA MODELS

- An entity is a thing or object in the real world that is distinguishable from other objects.
  - Rectangles represent entities
  - Diamonds represent relationship among entities.
  - Ellipse represent attributes
  - Lines represent link of attributes to entities to relationships.




Example of schema in the entity-relationship model

# DATA MODELS

Object based data models

- It is based on object oriented programming language paradigm.
- Inheritance, object identity and encapsulations
- It can be seen as extending the E-R model with opps concepts.
- Semi structured data models
- Semi structured data models permit the specification of data where individual data items of same type may have different set of attributes.
- XML language is widely used to represent semi structured data

# DATABASE LANGUAGES

- Data definition language- to define the data in the database
- Data Manipulation language- to manipulate the data in the database
- DDL:Specification notation for defining the database schema
- DDL is used to create the database, alter database and delete database.

Example:

```
create table account (
    account_number    char(10),
    branch_name       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)
  - DDL is used by conceptual schema
  - The internal DDL or also known as Data *storage and definition* language specifies the storage structure and access methods used DDl commands are Create, Alter and Drop only.

# DATABASE LANGUAGES

DML:

- Language for accessing and manipulating the data organized by the appropriate data model
  - DML also known as query language
  - DML is used to retrieve data from database, insertions of new data into database ,deletion or modification of existing data.
- Two classes of languages
  - Procedural – user specifies what data is required and how to get those data
  - Declarative (nonprocedural) – user specifies what data is required without specifying how to get those data
- SQL is the most widely used query language

# DATABASE USERS

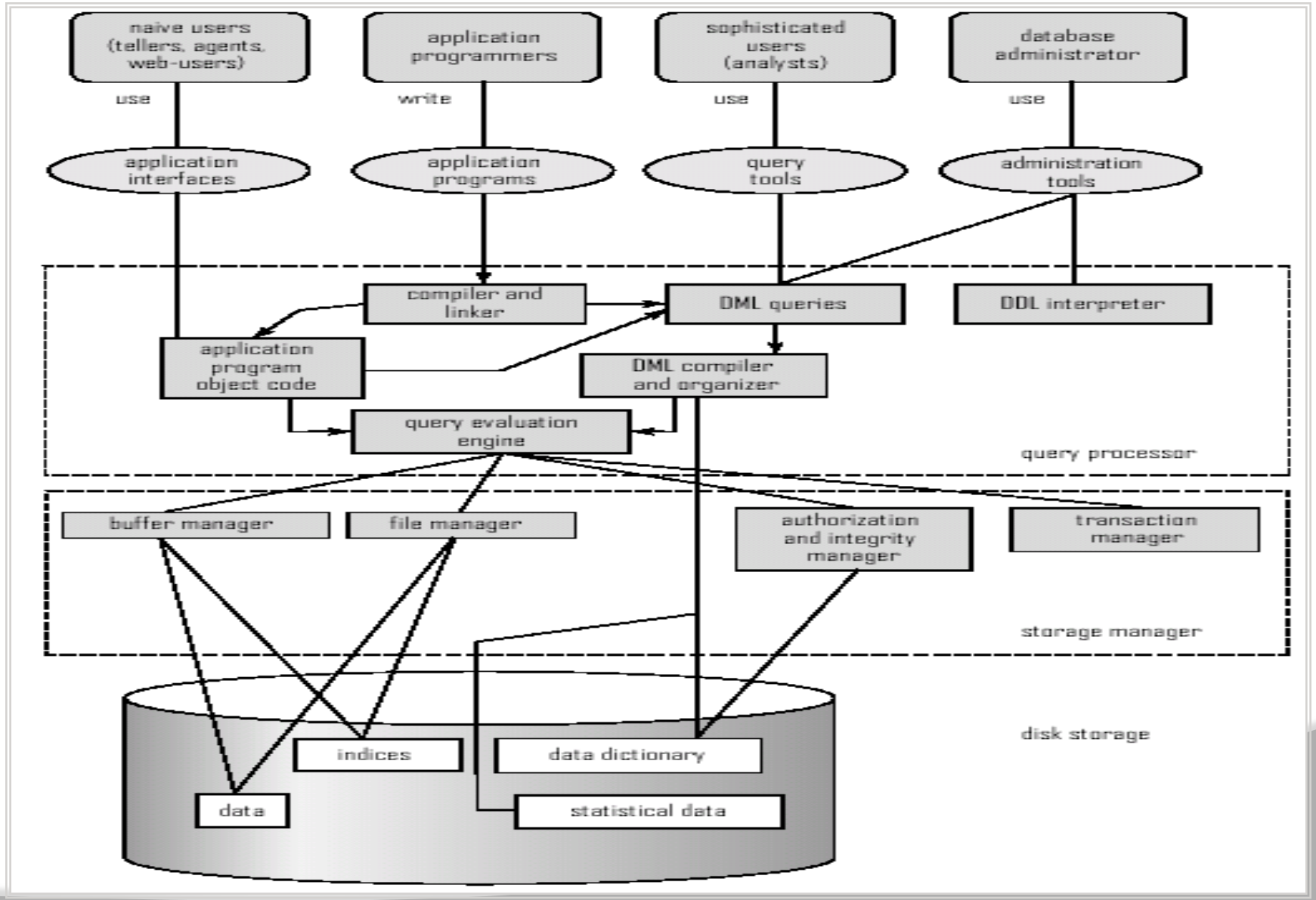
Users are differentiated by the way they expect to interact with the system

- Application programmers –are computer professionals who write appn prgms. They use RAD tools to construct forms and reports with minimum programming effect.
- Sophisticated users – interact with the system without writing programs, instead they form their requests in a database query language
- Specialized users – write specialized database applications that do not fit into the traditional data processing framework
- Ex: Computer aided design systems, knowledgebase expert systems.
- Naïve users – invoke one of the permanent application programs that have been written previously Examples, people accessing database over the web, bank tellers, clerical staff

# DATABASE USERS

- Database Administrator
  - Has central control of both data and programs to access that data.
  - Coordinates all the activities of the database system
    - has a good understanding of the enterprise's information resources and needs.
  - Database administrator's duties include:
    - Storage structure and access method definition
    - Schema and physical organization modification
    - Granting users authority to access the database
    - Backing up data
    - Monitoring performance and responding to changes
    - Periodically backing up the database, either on tapes or onto remote servers.

# VARIOUS COMPONENTS OF OVERALL DBS ARCHITECTURE



# VARIOUS COMPONENTS OF OVERALL DBS ARCHITECTURE

- Database access from application programs

- To access db, DML stmts need to be executed from host lang.

- 2 ways-
- by providing appn prgm interface that can b used to send
- DML and DDL stmts to database and retrieve results.
- Ex:ODBC & JDBC

- By extending host language syntax to embed DML calls
- within the host lang prgm.
- Data storage and Querying
- Storage management
- Query processing
- Transaction processing

# VARIOUS COMPONENTS OF OVERALL DBS ARCHITECTURE

- Storage manager is a program module that provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system.
- The storage manager is responsible to the following tasks:
  Interaction with the file manager
  Efficient storing, retrieving and updating of data
- Storage manager implements several data structures such as Data files, Data dictionary, Indices.

# VARIOUS COMPONENTS OF OVERALL DBS ARCHITECTURE

- Authorization and integrity manager tests for satisfaction of integrity constraints and checks the authority of users to access the data
- Transaction manager ensures database remains in consistent state despite system failures and concurrent transaction executions proceed without conflicting
- File manager manages allocation of space on disk storage and the data structures used to represent data on disk
- Buffer manager which is responsible for fetching data from disk storage into main memory and deciding what data to cache in main memory

# VARIOUS CONCEPTS OF ER MODEL

- *Entity:* Real-world object distinguishable from other objects.
- An entity is described (in DB) using a set of *attributes*.
- *Entity Set*: A collection of similar entities.
- E.g., all employees.
  - All entities in an entity set have the same set of attributes.
  - Each entity set has a *key*.(minimal set of attributes whose values uniquely identify entity in set)
  - Each attribute has a *domain*.
  - An entity is represented by a set of attributes, that is descriptive properties possessed by all members of an entity set.

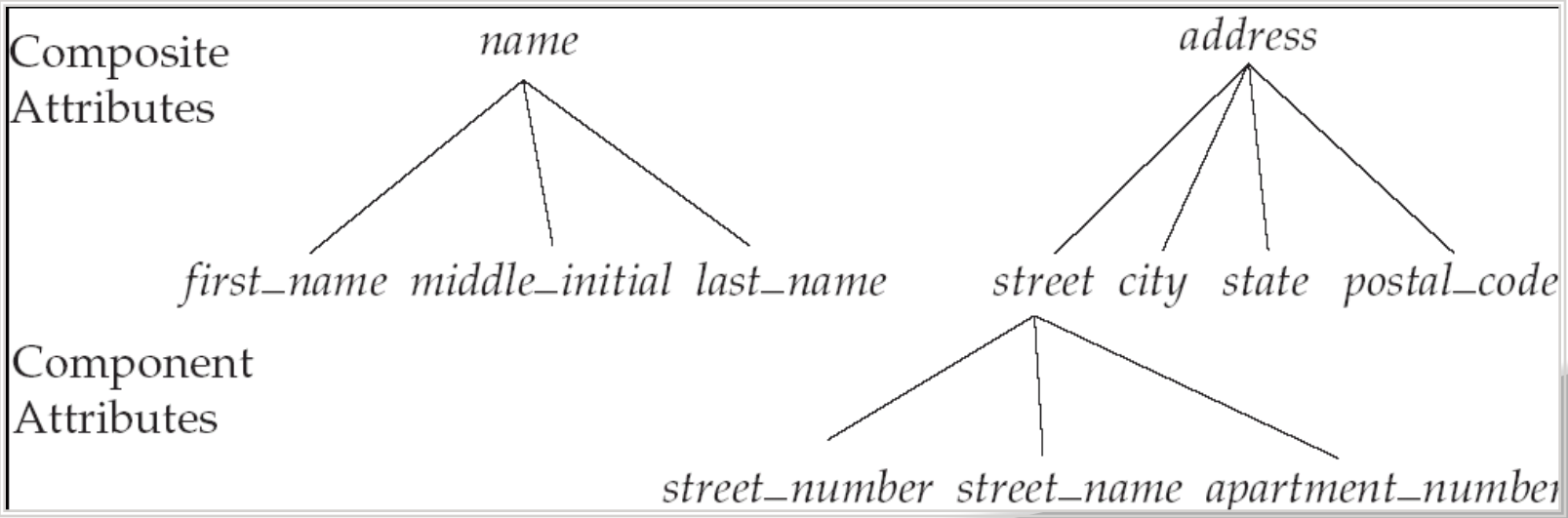
Example:

*customer* = (*customer_id, customer_name, customer_street, customer_city* )

*loan* = (*loan_number, amount* )

# VARIOUS CONCEPTS OF ER MODEL

- Domain – the set of permitted values for each attribute
- Attribute types:
    - *Simple* and *composite* attributes.
    - *Single-valued* and *multi-valued* attributes
        - Example: multivalued attribute: *phone_numbers*
    - *Derived* attributes
        - Example: age, given date_of_birth



# VARIOUS CONCEPTS OF ER MODEL

- *Relationship*: Association among two or more entities.
- E.g., Attishoo works in Pharmacy department.
- *Relationship Set*: Collection of similar relationships.
- syntax {(e1,…e2)|e1ЄE1, e2ЄE2….. enЄEn}
- A relationship is an association among several entities

  Example:

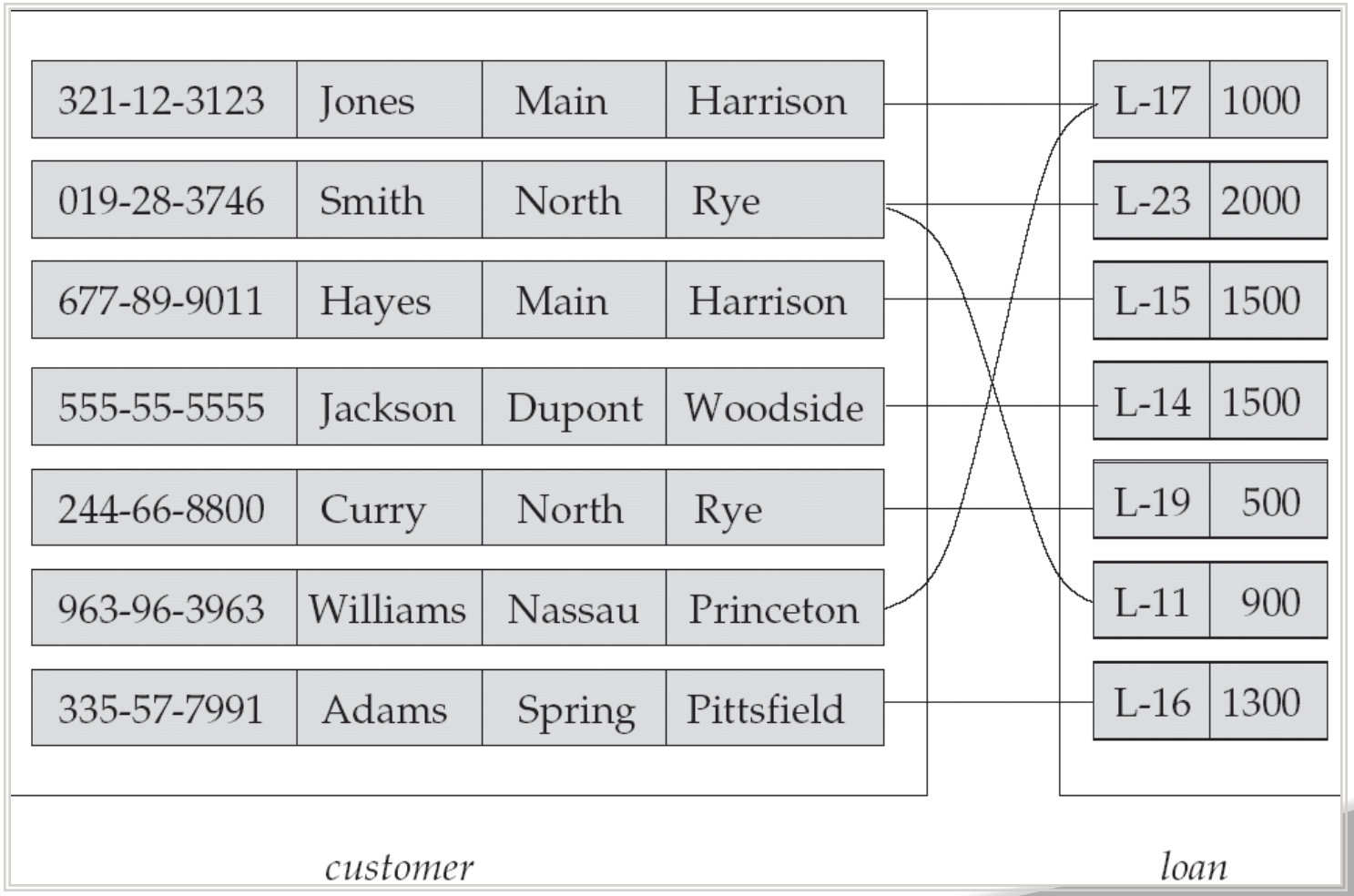
  Hayes *depositor* A-102

  *customer* entity relationshipset *account*
- A relationship set is a mathematical relation among $n \geq 2$ entities, each taken from entity sets

  $\{(e_1, e_2, \ldots e_n) \mid e_1 \in E_1, e_2 \in E_2, \ldots, e_n \in E_n\}$

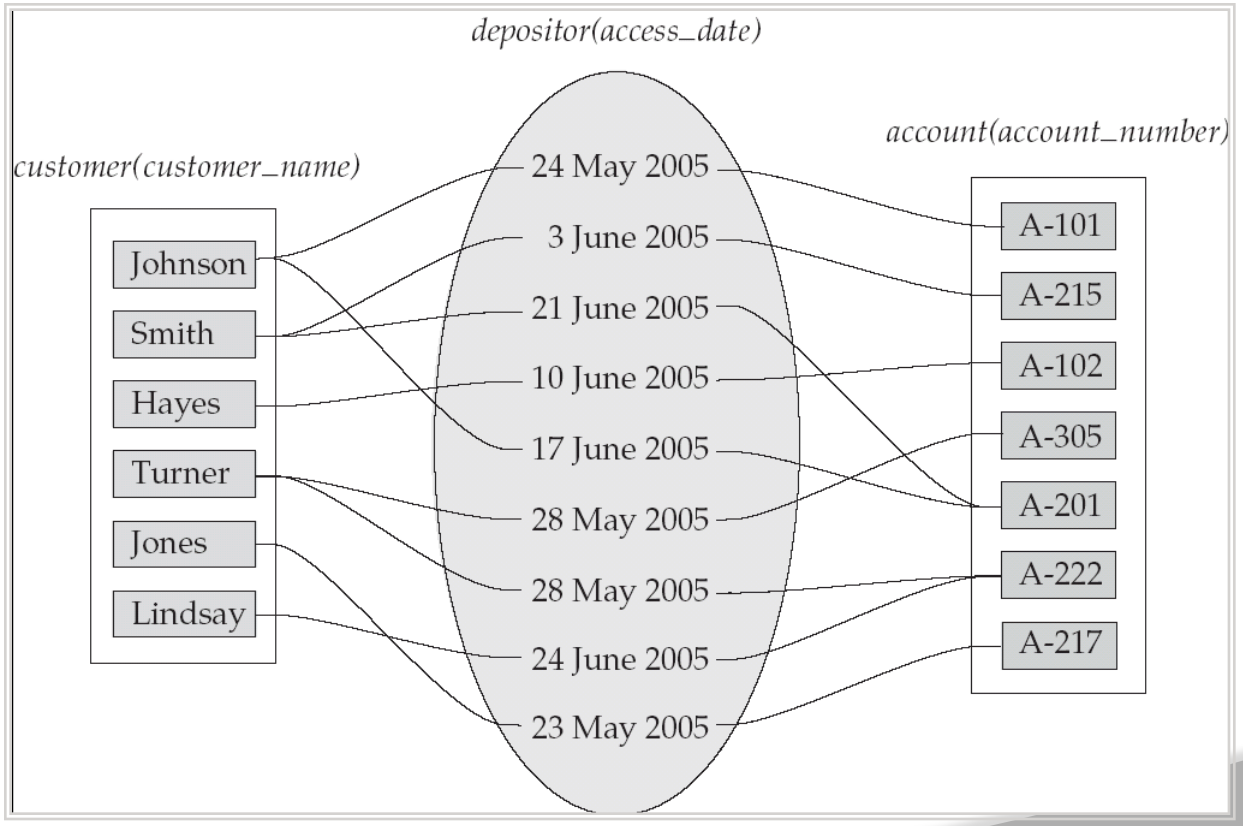
  where $(e_1, e_2, \ldots, e_n)$ is a relationship
- Example: (Hayes, A-102) $\in$ *depositor*

## Relationship Set *borrower*



# VARIOUS CONCEPTS OF ER MODEL

- An attribute can also be property of a relationship set.
- For instance, the *depositor* relationship set between entity sets *customer* and *account* may have the attribute *access-date*

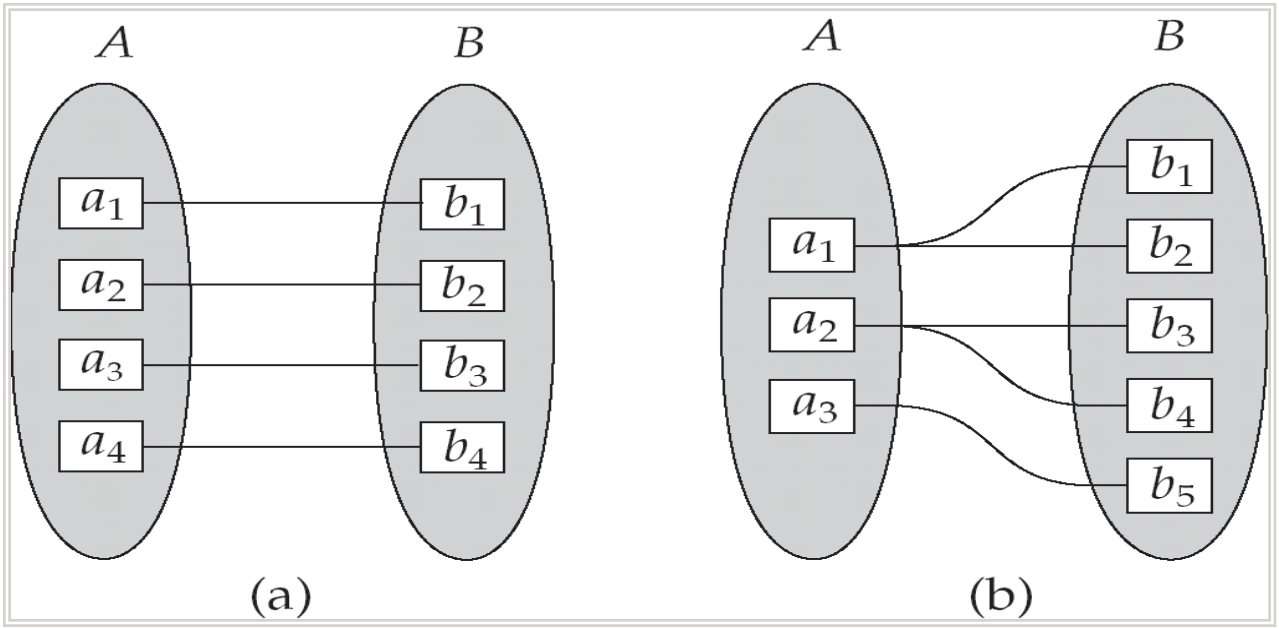


Degree of a Relationship Set

- Refers to number of entity sets that participate in a relationship set.
- Relationship sets that involve two entity sets are binary (or degree two).
- Relationship sets may involve more than two entity sets.

Mapping Cardinalities

# VARIOUS CONCEPTS OF ER MODEL

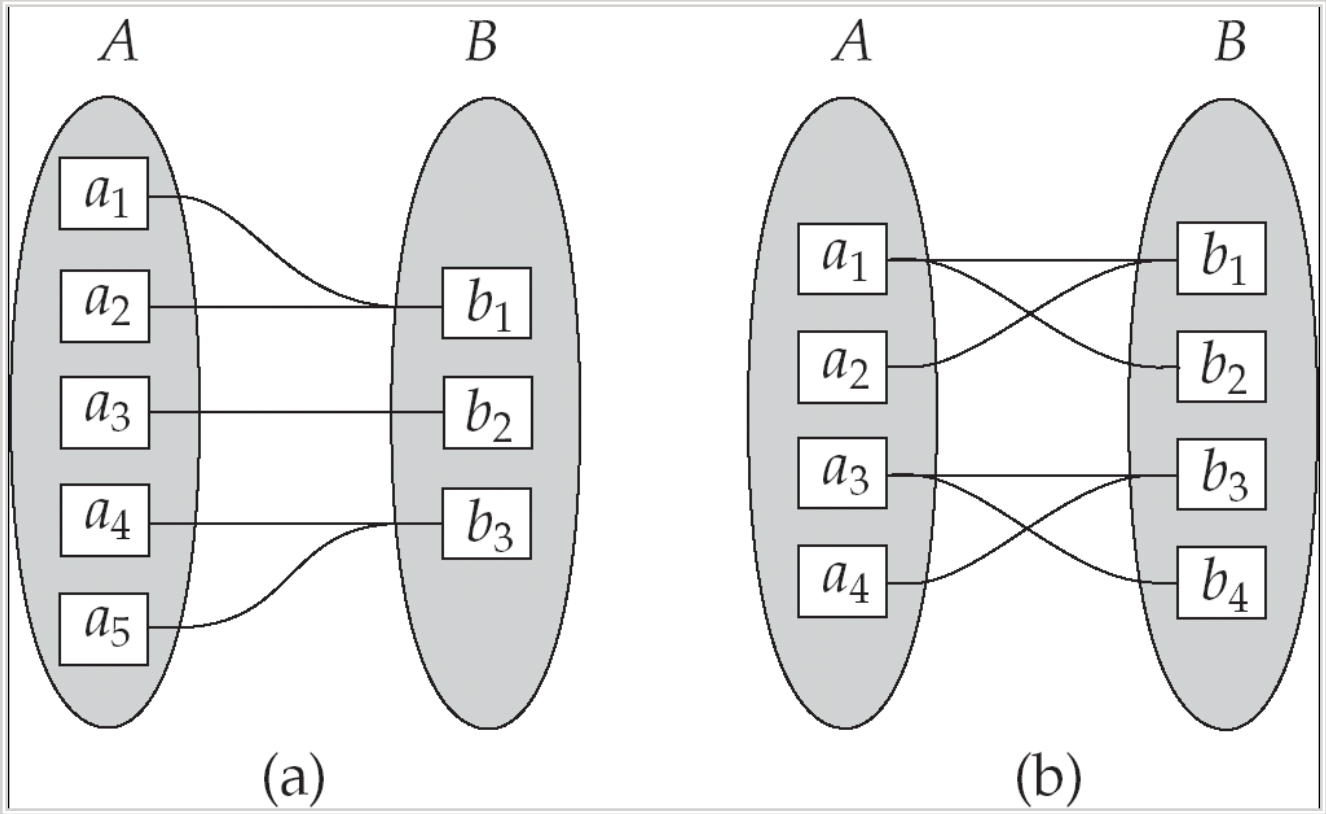



One to one

One to many

Note: Some elements in *A* and *B* may not be mapped to any elements in the other set

# VARIOUS CONCEPTS OF ER MODEL




Many to one

Many to many

Note: Some elements in A and B may not be mapped to any elements in the other set

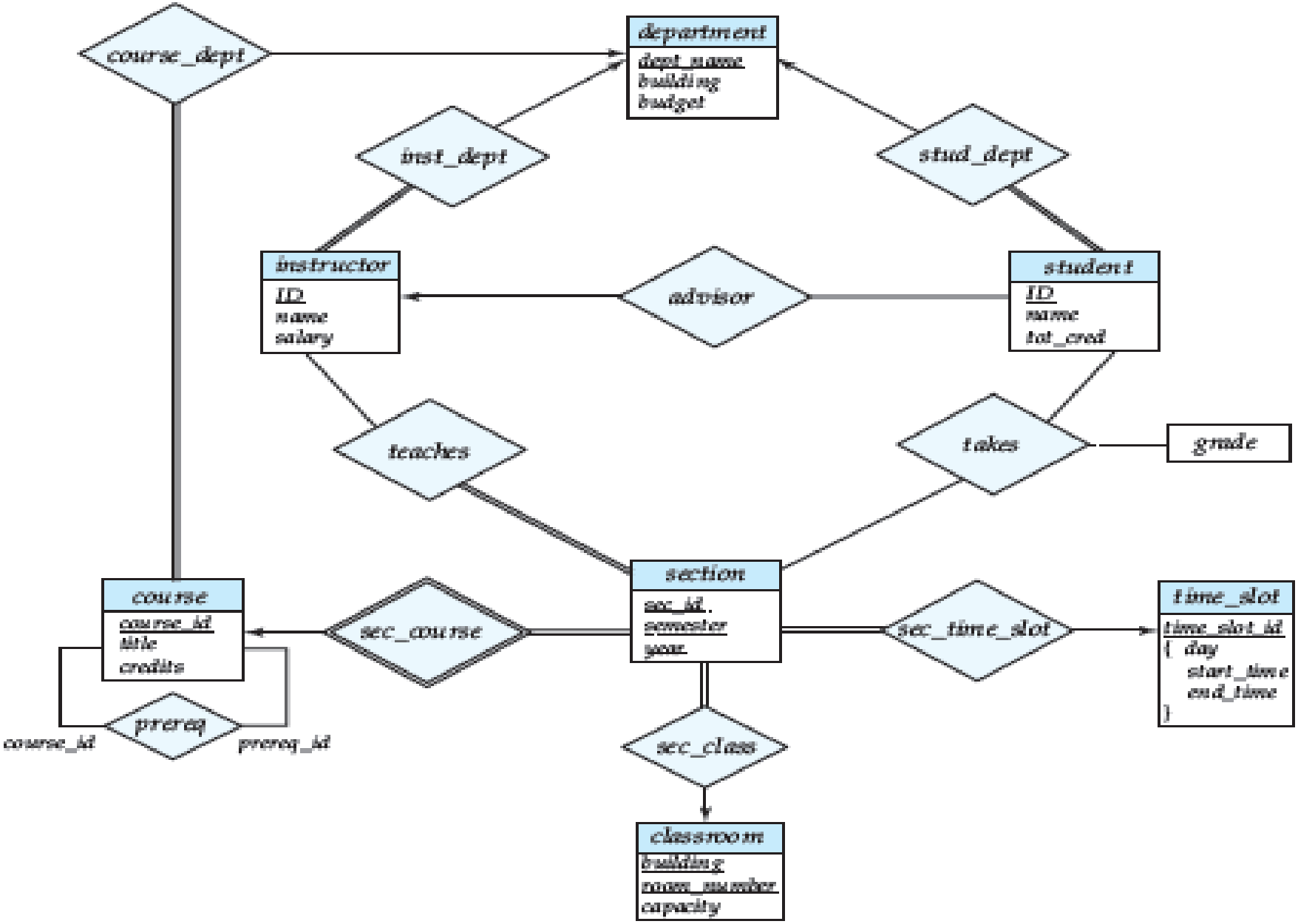
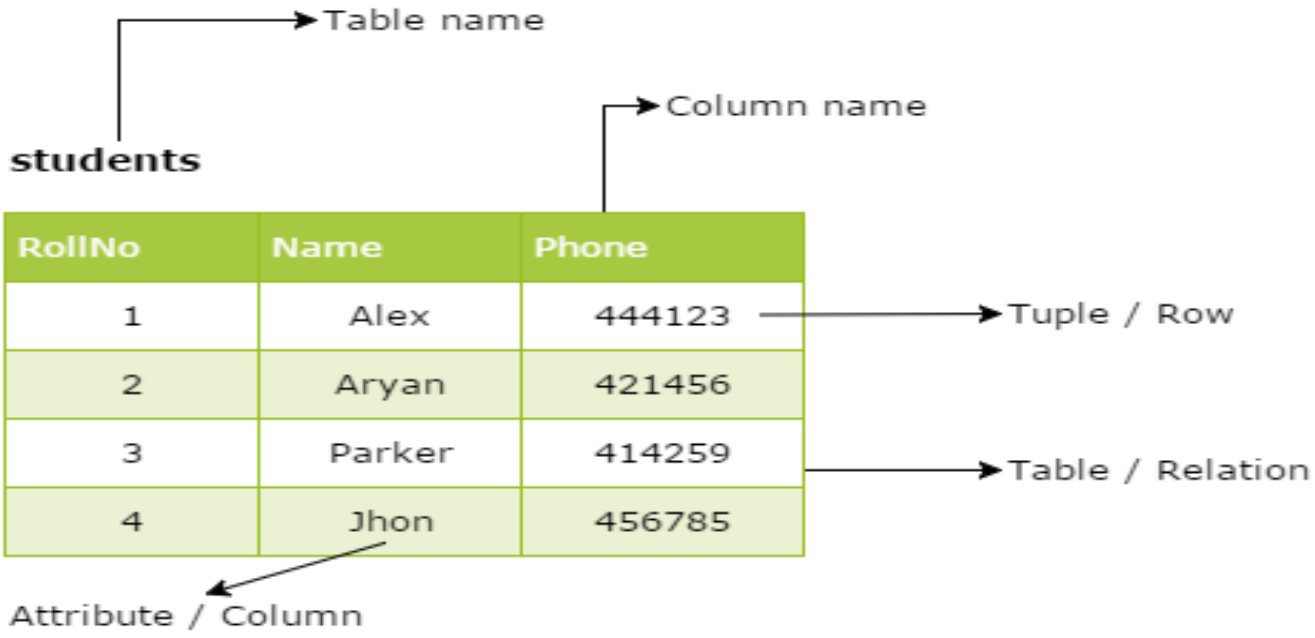



**Figure 7.15** E-R diagram for a university enterprise.

# RELATIONAL MODEL

- The relational model uses a collection of tables to represent both data and the relationships among those data. Each table has multiple columns, and each column has a unique name. Tables are also known as **relations.**
- **The relational model is an example of a record-based model.**
- In the relational model the term
- **Relation**: A relation is a table with columns and rows.
- **Tuple:** Each row in the relation is known as tuple.
- **attribute** :Each column in a table.
- **Relation Instance:** The set of tuples of a relation at a particular instance of time is called as relation instance.
- **Relation Schema:** A relation schema represents name of the relation with its attributes. e.g.; STUDENT (ROLL_NO, NAME, ADDRESS, PHONE and AGE) is relation schema for STUDENT.

- **Degree:** The number of attributes in the relation is known as degree of the relation.
- **Cardinality:** The number of tuples in a relation is known as cardinality.
- **Column:** Column represents the set of values for a particular attribute



- Constraints
- Every relation has some conditions that must hold for it to be a valid relation. These conditions are called **Relational Integrity Constraints**. There are three main integrity constraints −
- Key constraints(Entity Constraints.)
- Domain constraints
- Referential integrity constraints

**Key constraints**

An attribute that can uniquely identify a tuple in a relation is called the key of the table. The value of the attribute for different tuples in the relation has to be unique.

**Example:**

In the given table, CustomerID is a key attribute of Customer Table. It is most likely to have a single key for one customer, CustomerID =1 is only for the CustomerName =" Google".

| CustomerID | CustomerName | Status |
|---|---|---|
| 1 | Google | Active |
| 2 | Amazon | Active |
| 3 | Apple | Inactive |

**Domain Constraints:**

- These are attribute level constraints. An attribute can only take values which lie inside the domain range.
- e.g,; (1)If a constrains AGE>0 is applied on STUDENT relation, inserting negative value of AGE will result in failure.
  (2)telephone numbers cannot contain a digit outside 0-9.

**Referential Integrity:**

- When one attribute of a relation can only take values from other attribute of same relation or any other relation, it is called referential integrity.
- Referential integrity constraints work on the concept of Foreign Keys.
- A foreign key is a key attribute of a relation that can be referred in other relation.

# RELATIONAL MODEL

- Let us suppose we have 2 relations In the above example, we have 2 relations, Customer and Billing.Tuple for CustomerID =1 is referenced twice in the relationBilling.
  So we know CustomerName=Google has billing amount $300

| CustomerID | CustomerName | Status |
|---|---|---|
| 1 | Google | Active |
| 2 | Amazon | Active |
| 3 | Apple | Inactive |

Customer

Billing

| InvoiceNo | CustomerID | Amount |
|---|---|---|
| 1 | 1 | $100 |
| 2 | 1 | $200 |
| 3 | 2 | $150 |

# RELATIONAL MODEL

- Entity-Set and Keys
- Key is an attribute or collection of attributes that uniquely identifies an entity among entity set.

  For example, the roll_number of a student makes him/her identifiable among students.
- **Super Key** − A set of attributes (one or more) taken collectively, allow us to identify uniquely a tuple in the relation.
- **Candidate Key** − A minimal super key is called a candidate key. An entity set may have more than one candidate key.
- **Primary Key** − A primary key is one of the candidate keys chosen by the database designer to uniquely identify the entity set.
- **Composite key** − If there is a combination of two or more attributes which is being used as the primary key then we call it as a composite key.

## Operations in Relational Model

Four basic update operations performed on relational database model are

- Insert, update, delete and select.
- Insert is used to insert data into the relation
- Delete is used to delete tuples from the table.
- Modify allows you to change the values of some attributes in existing tuples.
- Select allows you to choose a specific range of data.

| CustomerID | CustomerName | Status |
|---|---|---|
| 1 | Google | Active |
| 2 | Amazon | Active |
| 3 | Apple | Inactive |


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

# RELATIONAL MODEL

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

UPDATE

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

| CustomerID | CustomerName | Status |
|---|---|---|
| 1 | Google | Active |
| 2 | Amazon | Active |
| 3 | Apple | Inactive |

INSERT

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

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

UPDATE

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

**Advantages of using Relational model**

- **Simplicity**: A relational data model is simpler than the hierarchical and network model.
- **Structural Independence**: The relational database is only concerned with data and not with a structure. This can improve the performance of the model.
- **Easy to use**: The relational model is easy as tables consisting of rows and columns is quite natural and simple to understand
- **Query capability**: It makes possible for a high-level query language like SQL to avoid complex database navigation.
- **Data independence**: The structure of a database can be changed without having to change any application.
- **Scalable**: Regarding a number of records, or rows, and the number of fields, a database should be enlarged to enhance its usability.

**Disadvantages of using Relational model**

- Few relational databases have limits on field lengths which can't be exceeded.
- Relational databases can sometimes become complex as the amount of data grows, and the relations between pieces of data become more complicated.
- Complex relational database systems may lead to isolated databases where the information cannot be shared from one system to another.


# MODULE II
# RELATIONAL APPROACH

| CLOs | Course Learning Outcome |
|---|---|
| CLO 6 | Determine Relational algebra, The Self variable. |
| CLO7 | Understand selection and projection, set operations. |
| CLO 8 | Determine renaming, joins, division. |
| CLO 9 | Use examples of algebra queries. |
| CLO 10 | Illustrate Tuple relational calculus, Domain relational calculus, and also expressive power of algebra and calculus |

# RELATIONAL ALGEBRA AND CALCULUS

- Two mathematical Query Languages form the basis for “real” languages (e.g. SQL), and for implementation:
  - Relational Algebra: More operational, very useful for representing execution plans.
  - Relational Calculus: Lets users describe what they want, rather than how to compute it. (Non-operational, declarative.)

# RELATIONAL ALGEBRA AND CALCULUS

- Basic operations:
  - Selection ( $\sigma$ ) Selects a subset of rows from relation.
  - Projection ( $\pi$ ) Deletes unwanted columns from relation.
  - Cross-product ( $\times$ ) Allows us to combine two relations.
  - Set-difference ( - ) Tuples in reln. 1, but not in reln. 2.
  - Union ( U ) Tuples in reln. 1 and in reln. 2.
- Additional operations:
  - Intersection, join, division, renaming

# PROJECTION

- *Schema* of result contains exactly the fields in the projection list, with the same names that they had in the (only) input relation.
- Projection operator has to eliminate *duplicates*! (Why??)
  - Note: real systems typically don't do duplicate elimination unless the user explicitly asks for it.

| sname | rating |
|---|---|
| yuppy | 9 |
| lubber | 8 |
| guppy | 5 |
| rusty | 10 |

$\pi_{sname,rating}(S2)$

| age |
|---|
| 35.0 |
| 55.5 |

$\pi_{age}(S2)$

# SELECTION

- Selects rows that satisfy *selection condition*.
- No duplicates in result! (Why?)
- *Schema* of result identical to schema of (only) input relation.
- *Result* relation can be the *input* for another relational algebra operation! (*Operator composition.*)

| sid | sname | rating | age |
|---|---|---|---|
| 28 | yuppy | 9 | 35.0 |
| 58 | rusty | 10 | 35.0 |

| sname | rating |
|---|---|
| yuppy | 9 |
| rusty | 10 |

$$\sigma_{rating>8}(S2) \qquad \pi_{sname,rating}(\sigma_{rating>8}(S2))$$

# SET OPERATIONS

- All of these operations take two input relations, which must be union-compatible: Same number of fields. `Corresponding' fields have the same type. What is the schema of result?

| sid | sname | rating | age |
|---|---|---|---|
| 22 | dustin | 7 | 45.0 |
| 31 | lubber | 8 | 55.5 |
| 58 | rusty | 10 | 35.0 |
| 44 | guppy | 5 | 35.0 |
| 28 | yuppy | 9 | 35.0 |

$S1 \cup S2$

| sid | sname | rating | age |
|---|---|---|---|
| 22 | dustin | 7 | 45.0 |

$S1 - S2$

| sid | sname | rating | age |
|---|---|---|---|
| 31 | lubber | 8 | 55.5 |
| 58 | rusty | 10 | 35.0 |

$S1 \cap S2$

# SET OPERATIONS

Cross product

- Each row of S1 is paired with each row of R1.
- *Result schema* has one field per field of S1 and R1, with field names `inherited' if possible.

  *Conflict*: Both S1 and R1 have a field called *sid*.

  S1 X R1

| (sid) | sname | rating | age | (sid) | bid | day |
|---|---|---|---|---|---|---|
| 22 | dustin | 7 | 45.0 | 22 | 101 | 10/10/96 |
| 22 | dustin | 7 | 45.0 | 58 | 103 | 11/12/96 |
| 31 | lubber | 8 | 55.5 | 22 | 101 | 10/10/96 |
| 31 | lubber | 8 | 55.5 | 58 | 103 | 11/12/96 |
| 58 | rusty | 10 | 35.0 | 22 | 101 | 10/10/96 |
| 58 | rusty | 10 | 35.0 | 58 | 103 | 11/12/96 |

# SET OPERATIONS

Cross product

- Each row of S1 is paired with each row of R1.
- *Result schema* has one field per field of S1 and R1, with field names `inherited' if possible.

  *Conflict*: Both S1 and R1 have a field called *sid*.

  S1 X R1

| (sid) | sname | rating | age | (sid) | bid | day |
|---|---|---|---|---|---|---|
| 22 | dustin | 7 | 45.0 | 22 | 101 | 10/10/96 |
| 22 | dustin | 7 | 45.0 | 58 | 103 | 11/12/96 |
| 31 | lubber | 8 | 55.5 | 22 | 101 | 10/10/96 |
| 31 | lubber | 8 | 55.5 | 58 | 103 | 11/12/96 |
| 58 | rusty | 10 | 35.0 | 22 | 101 | 10/10/96 |
| 58 | rusty | 10 | 35.0 | 58 | 103 | 11/12/96 |

# RENAMING

- *Renaming operator(*ρ):

_ρ (old name -> new name)  or
_ρ (position -> new name)

$$\rho\,(C(1 \rightarrow sid1, 5 \rightarrow sid2),\ S1 \times R1)$$

# JOINS

- *Condition Join*: $R \bowtie_c S = \sigma_c (R \times S)$

$$S1 \bowtie_{S1.sid < R1.sid} R1$$

| (sid) | sname | rating | age | (sid) | bid | day |
|---|---|---|---|---|---|---|
| 22 | dustin | 7 | 45.0 | 58 | 103 | 11/12/96 |
| 31 | lubber | 8 | 55.5 | 58 | 103 | 11/12/96 |

- *Result schema* same as that of cross-product.
- Fewer tuples than cross-product, might be able to compute more efficiently
- Sometimes called a *theta-join*.

# JOINS

*Equi-Join*: A special case of condition join where the condition *c* contains only *equalities*.

$$S1 \bowtie_{sid} R1$$

| sid | sname | rating | age | bid | day |
|---|---|---|---|---|---|
| 22 | dustin | 7 | 45.0 | 101 | 10/10/96 |
| 58 | rusty | 10 | 35.0 | 103 | 11/12/96 |

- *Result schema* similar to cross-product, but only one copy of fields for which equality is specified.
- *Natural Join*: Equijoin on *all* common fields.
- If two relations have no attributes in common, natural join is simply cross product.

# DIVISION

- Not supported as a primitive operator, but useful for expressing queries like:
  *Find sailors who have reserved all boats*.
- Let *A* have 2 fields, *x* and *y*; *B* have only field *y*:
  - *A/B* =
  - i.e., *A/B* contains all *x* tuples (sailors) such that for *every* *y* tuple (boat) in *B*, there is an *xy* tuple in *A*.
  - *Or*: If the set of *y* values (boats) associated with an *x* value (sailor) in *A* contains all *y* values in *B*, the *x* value is in *A/B*.
- In general, *x* and *y* can be any lists of fields; *y* is the list of fields in *B*, and *x* *y* is the list of fields of *A*.

# DIVISION

Examples of Division A/B

| sno | pno |
|---|---|
| s1 | p1 |
| s1 | p2 |
| s1 | p3 |
| s1 | p4 |
| s2 | p1 |
| s2 | p2 |
| s3 | p2 |
| s4 | p2 |
| s4 | p4 |

*A*

| pno |
|---|
| p2 |

*B1*

| sno |
|---|
| s1 |
| s2 |
| s3 |
| s4 |

*A/B1*

| pno |
|---|
| p2 |
| p4 |

*B2*

| sno |
|---|
| s1 |
| s4 |

*A/B2*

| pno |
|---|
| p1 |
| p2 |
| p4 |

*B3*

| sno |
|---|
| s1 |

*A/B3*

# RELATIONAL CALUCLUS

- Comes in two flavors: *Tuple relational calculus* (TRC) and *Domain relational calculus* (DRC).
- Calculus has *variables, constants, comparison ops*, *logical connectives* and *quantifiers*.
  - *TRC*: Variables range over (i.e., get bound to) *tuples*.
  - *DRC*: Variables range over *domain elements* (= field values).
  - Both TRC and DRC are simple subsets of first-order logic.
- Expressions in the calculus are called *formulas*. An answer tuple is essentially an assignment of constants to variables that make the formula evaluate to *true*.

# TUPLE RELATIONAL CALUCLUS

- A tuple rc query has the form {T|P(T)} where T is a tuple variable and P(T) denotes a formula that describes T.

- Find all sailors with rating above 7
- {S|S € Sailors Л s.rating>7}

- Let Rel be a relation name, R & S be tuple variables,'a' be an attribute of R and 'b' be attribute of S. Let op denote operator.
- An atomic formula is one of the following
- R € Rel, R.a € S.b, R.a op constant or constant op R.a

# TUPLE RELATIONAL CALUCLUS

- A formula is recursively defined to be one of the following any atomic formula
- ┐P,PЛQ,P V Q or P=>Q
- эR(P(R)) where R is tuple variable
- forall R(P(R)) where R is tuple variable
- A variable is said to be free in formula if it does not contain an occurence of quantifiers that bind it.

- Find the names and ages of sailors with rating above 7

{P| эS є Sailors(S.Rating >7 Л P.name=S.Sname Л P.age=S.age)

# DOMAIN RELATIONAL CALUCLUS

1. In the tuple relational calculus, you have use variables that have series of tuples in a relation.
2. In the domain relational calculus, you will also use variables, but in this case, the variables take their values from domains of attributes rather than tuples of relations.
3. A domain relational calculus expression has the following general format:
   $\{d1, d2, \ldots, dn \mid F(d1, d2, \ldots, dm)\}$ $m \geq n$
   where d1, d2, . . . , dn, . . . , dm stand for domain variables and F(d1, d2, . . . , dm) stands for a formula composed of atoms.

# DOMAIN RELATIONAL CALUCLUS

- Example:
  select TCHR_ID and TCHR_NAME of teachers who work for department 8, (where suppose - dept. 8 is Computer Application Department)
- {<tchr_id, tchr_name=""> | <tchr_id, tchr_name=""> ? TEACHER Λ DEPT_ID = 10}
- Get the name of the department name where Karlos works:
  {DEPT_NAME |< DEPT_NAME > ? DEPT Λ ? DEPT_ID ( ? TEACHER Λ TCHR_NAME = Karlos)}
- It is to be noted that these queries are safe. The use domain relational calculus is restricted to safe expressions; moreover, it is equivalent to the tuple relational calculus which in turn is similar to the relational algebra.

# EXPRESSIVE POWER OF ALGEBRA AND CALCULUS

- Regarding expressiveness, we can show that every query that can be expressed using a *safe relational calculus query* can also be expressed as a relational algebra query.

- The expressive power of relational algebra is often used as a metric of how powerful a relational database query language is.

- If a query language can express all the queries that we can express in relational algebra, it is said to be relationally complete.

- A practical query language is expected to be relationally complete; in addition, commercial query languages typically support features that allow us to express some queries that cannot be expressed in relational algebra.


# MODULE III
# SQL QUERY - BASICS , RDBMS - NORMALIZATION

| CLOs | Course Learning Outcome |
|---|---|
| CLO 11 | Understand SQL – Data Definition commands, Queries with various options. |
| CLO 12 | Analyze the concept of Mata manipulation commands, Views, Joins, views. |
| CLO 13 | Illustrate Calling a function, Returning multiple values from a function. |
| CLO 14 | Contrast the Usage of Relational database design, Functional dependencies, Armstrong Axioms |
| CLO 15 | Define Normalization, 2nd and 3rd Normalization, Basic definitions of MVDs and JDs, 4th and 5th normal forms |

# SQL – DATA DEFINITION COMMANDS

- An SQL relation is defined using the create table command:
- create table $r$ ($A_1$ $D_1$, $A_2$ $D_2$, ..., $A_n$ $D_n$,
  (integrity-constraint$_1$),
  ...,(integrity-constraint$_k$))
  - $r$ is the name of the relation
  - each $A_i$ is an attribute name in the schema of relation $r$
  - $D_i$ is the data type of attribute $A_i$

  Example:

  create table *branch*
  (*branch_name* char(15),
  *branch_city* char(30),
  *assets* integer)

# SQL – DATA DEFINITION COMMANDS

- **char(n).** Fixed length character string, with user-specified length *n*.
- **varchar(n).** Variable length character strings, with user-specified maximum length *n.*
- **int.** Integer (a finite subset of the integers that is machine-dependent).
- **smallint.** Small integer (a machine-dependent subset of the integer domain type).
- **numeric(p,d).** Fixed point number, with user-specified precision of *p* digits, with *n* digits to the right of decimal point.
- **float(n).** Floating point number, with user-specified precision of at least *n* digits.

# SQL – DATA DEFINITION COMMANDS

- The drop table command deletes all information about the dropped relation from the database.
- The alter table command is used to add attributes to an existing relation:

  alter table *r* add *A D*

  where *A* is the name of the attribute to be added to relation *r* and *D* is the domain of *A.*
  - All tuples in the relation are assigned *null* as the value for the new attribute.
- The alter table command can also be used to drop attributes of a relation:

  alter table *r* drop *A*

  where *A* is the name of an attribute of relation *r*
  - Dropping of attributes not supported by many databases

# SQL – DATA DEFINITION COMMANDS

- A typical SQL query has the form:

  select $A_1, A_2, \ldots, A_n$
  from $r_1, r_2, \ldots, r_m$
  where

  - $A_i$ represents an attribute
  - $R_i$ represents a relation
  - $P$ is a predicate.
- This query is equivalent to the relational algebra expression.
- The result of an SQL query is a relation.

$$\Pi_{A_1,A_2,\ldots,A_n}(\sigma_P(r_1 \times r_2 \times \ldots \times r_m))$$

# SQL – DATA DEFINITION COMMANDS

- The select clause list the attributes desired in the result of a query
  - corresponds to the projection operation of the relational algebra
- Example: find the names of all branches in the *loan* relation:

  select *branch_name*
  from *loan*
- In the relational algebra, the query would be:

  $\Pi_{branch\_name}$ (*loan*)
- NOTE: SQL names are case insensitive (i.e., you may use upper- or lower-case letters.)
  - E.g. *Branch_Name* ≡ *BRANCH_NAME* ≡ *branch_name*
  - Some people use upper case wherever we use bold font.

# SQL – DATA DEFINITION COMMANDS

- SQL allows duplicates in relations as well as in query results.
- To force the elimination of duplicates, insert the keyword distinct after select.
- Find the names of all branches in the *loan* relations, and remove duplicates
- select distinct *branch_name* from *loan*
- The keyword all specifies that duplicates not be removed.

select all *branch_name* from *loan*

# SQL – DATA DEFINITION COMMANDS

- The where clause specifies conditions that the result must satisfy
  - Corresponds to the selection predicate of the relational algebra.
- To find all loan number for loans made at the Perryridge branch with loan amounts greater than $1200.

```
select loan_number
from loan
where branch_name = 'Perryridge'
 and amount > 1200
```

- Comparison results can be combined using the logical connectives and, or, and not.

# QUERIES WITH VARIOUS OPTIONS

- The SQL CREATE TABLE Statement
- The CREATE TABLE statement is used to create a new table in a database.

Syntax

CREATE TABLE *table_name* (
*column1 datatype*,
*column2 datatype*,
*column3 datatype*, ....);

# QUERIES WITH VARIOUS OPTIONS

- Example

```
CREATE TABLE Persons (
PersonID int NOT NULL,
LastName varchar(255),
FirstName varchar(255),
Address varchar(255),
City varchar(255) );
```

- The PersonID column is of type int and will hold an integer.
- The LastName, FirstName, Address, and City columns are of type varchar and will hold characters, and the maximum length for these fields is 255 chars

# QUERIES WITH VARIOUS OPTIONS

- 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
- CREATE TABLE Persons (
  ID int NOT NULL,
  LastName varchar(255) NOT NULL,
  FirstName varchar(255),
  Age int,
  City varchar(255) DEFAULT 'Sandnes');

# QUERIES WITH VARIOUS OPTIONS

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

Example

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

# QUERIES WITH VARIOUS OPTIONS

- **ALTER:** It is used to alter the structure of the database
- To add a new column in the table
- Sql>ALTER TABLE table_name ADD column_name COLUMN-definition;
- To modify existing column in the table:
- ALTER TABLE MODIFY(COLUMN DEFINITION....);

EXAMPLE

Sql>ALTER TABLE STU_DETAILS ADD(ADDRESS VARCHAR2(20));

Sql>ALTER TABLE STU_DETAILS MODIFY (NAME VARCHAR2(20));

# QUERIES WITH VARIOUS OPTIONS

- **TRUNCATE**: It is used to delete all the rows from the table and free the space containing the table.

Syntax:

- TRUNCATE TABLE table_name;

Example:

- TRUNCATE TABLE EMPLOYEE;
- The **DROP statement** destroys the objects like an existing database, table, index, or view

•DROP TABLE table_name;

•DROP TABLE STU_DETAILS

# DATA MANIPULATION COMMANDS

- **INSERT:** The INSERT statement is a SQL query. It is used to insert data into the row of a table.

  **Syntax:**

  INSERT INTO TABLE_NAME
  VALUES (value1, value2, value3, .... valueN);

  **example:**

  INSERT INTO student VALUES ("Sdname", "DBMS");

- **UPDATE:** This command is used to update or modify the value of a column in the table.

  **Syntax:**

  UPDATE table_name SET [column_name1= value1,...
  column_nameN = valueN] [WHERE CONDITION]

  **example:**

  UPDATE students SET User_Name = 'Sdname' WHERE Stud
  ent_Id = '3'

# DATA MANIPULATION COMMANDS

- **DELETE:** It is used to remove one or more row from a table.

  **Syntax:**

  DELETE FROM table_name [WHERE condition];

  **For example:**

  DELETE FROM student  WHERE  sname=“abc";

- **SELECT:**  It is used to select the attribute based on the condition described by WHERE clause.

  **Syntax:**

  SELECT expressions FROM TABLES

  WHERE conditions;

  **For example:**

  SELECT emp_name  FROM employee

  WHERE age > 20;

  SELECT * FROM emp

# VIEWS

- A view is nothing more than a SQL statement that is stored in the database with an associated name.
- A view is actually a composition of a table in the form of a predefined SQL query.
- A view can contain all rows of a table or select rows from a table.
- A view can be created from one or many tables which depends on the written SQL query to create a view.

Views, which are a type of virtual tables allow users to do the following –

- Structure data in a way that users or classes of users find natural or intuitive.
- Restrict access to the data in such a way that a user can see and (sometimes) modify exactly what they need and no more.
- Summarize data from various tables which can be used to generate reports.

# VIEWS

- To create a view, a user must have the appropriate system privilege according to the specific implementation.
- The basic CREATE VIEW syntax is as follows −

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

- You can include multiple tables in your SELECT statement in a similar way as you use them in a normal SQL SELECT query.

# VIEWS

- Following is an example to create a view from the CUSTOMERS table. This view would be used to have customer name and age from the CUSTOMERS table.

```
SQL > CREATE VIEW CUSTOMERS_VIEW AS
SELECT name, age
FROM  CUSTOMERS;
```

- Now, you can query CUSTOMERS_VIEW in a similar way as you query an actual table. Following is an example for the same.

```
SQL > SELECT * FROM CUSTOMERS_VIEW;
```

# VIEWS

- Updating a View
- A view can be updated under certain conditions which are given below –
  - The SELECT clause may not contain the keyword DISTINCT.
  - The SELECT clause may not contain summary functions.
  - The SELECT clause may not contain set functions.
  - The SELECT clause may not contain set operators.
  - The SELECT clause may not contain an ORDER BY clause.
  - The FROM clause may not contain multiple tables.
  - The WHERE clause may not contain subqueries.
  - The query may not contain GROUP BY or HAVING.
  - Calculated columns may not be updated.
  - All NOT NULL columns from the base table must be included in the view in order for the INSERT query to function.

# VIEWS

- Inserting Rows into a View
  - Rows of data can be inserted into a view.
  - The same rules that apply to the UPDATE command also apply to the INSERT command.
  - Here, we cannot insert rows in the CUSTOMERS_VIEW because we have not included all the NOT NULL columns in this view, otherwise you can insert rows in a view in a similar way as you insert them in a table.
- Deleting Rows into a View

  Rows of data can be deleted from a view. The same rules that apply to the UPDATE and INSERT commands apply to the DELETE command.
- Following is an example to delete a record having AGE = 22.

```
SQL > DELETE FROM CUSTOMERS_VIEW
WHERE age = 22;
```

# VIEWS

- Dropping Views

  Obviously, where you have a view, you need a way to drop the view if it is no longer needed.
- The syntax is DROP VIEW view_name;
- Following is an example to drop the CUSTOMERS_VIEW from the CUSTOMERS table.

  DROP VIEW CUSTOMERS_VIEW;

# JOINS

- Join operations take two relations and return as a result another relation.
- These additional operations are typically used as subquery expressions in the from clause
- Join condition – defines which tuples in the two relations match, and what attributes are present in the result of the join.
- Join type – defines how tuples in each relation that do not match any tuple in the other relation (based on the join condition) are treated.

| *Join types* |
| --- |
| **inner join** |
| **left outer join** |
| **right outer join** |
| **full outer join** |

| *Join Conditions* |
| --- |
| **natural** |
| **on** <predicate> |
| **using** $(A_1, A_1, \ldots, A_n)$ |

# JOINS

- Relation *loan*

| *loan_number* | *branch_name* | *amount* |
|---|---|---|
| L-170 | Downtown | 3000 |
| L-230 | Redwood | 4000 |
| L-260 | Perryridge | 1700 |

*loan*

- Relation *borrower*

| *customer_name* | *loan_number* |
|---|---|
| Jones | L-170 |
| Smith | L-230 |
| Hayes | L-155 |

*borrower*

- *Select S.sid, R.bid from Sailors S natural left outer join Reserves R*
  - *loan* **inner join** *borrower* **on** *loan.loan_number = borrower.loan_number*

| Sid | Bid |
|---|---|
| 22 | 101 |
| 31 | Null |
| 58 | 103 |

# INTEGRITY

- Data integrity in the database is the correctness, consistency and completeness of data.
- Data integrity is enforced using the following three integrity constraints:
- **Entity Integrity** - This is related to the concept of primary keys. All tables should have their own primary keys which should uniquely identify a row and not be NULL.
- **Referential Integrity** - This is related to the concept of foreign keys. A foreign key is a key of a relation that is referred in another relation.
- **Domain Integrity** - This means that there should be a defined domain for all the columns in a database.DB2 database and functions can be managed by two different modes of security controls: Authentication, Authorization

# INTEGRITY

**Authentication**

- Authentication is the process of confirming that a user logs in only in accordance with the rights to perform the activities he is authorized to perform.
- User authentication can be performed at operating system level or database level itself.
- By using authentication tools for biometrics such as retina and figure prints are in use to keep the database from hackers or malicious users.

**Authorization**

- You can access the DB2 Database and its functionality within the DB2 database system, which is managed by the DB2 Database manager.
- The manager obtains information about the current authenticated user, that indicates which database operation the user can perform or access.

# SECURITY

- DB2 tables and configuration files are used to record the permissions associated with authorization names. When a user tries to access the data, the recorded permissions verify the following permissions:
    - Authorization name of the user
    - Which group belongs to the user
    - Which roles are granted directly to the user or indirectly to a group
    - Permissions acquired through a trusted context.

# SECURITY

- While working with the SQL statements, the DB2 authorization model considers the combination of the following permissions:
  - Permissions granted to the primary authorization ID associated with the SQL statements.
  - Secondary authorization IDs associated with the SQL statements.
  - Granted to PUBLIC
  - Granted to the trusted context role.
  - Instance level authorities

# PITFALLS OF RDBD

- Obviously, we can have good and bad designs.
- Among the undesirable design items are:
    - Repetition of information
    - Inability to represent certain information
- The relation *lending* with the schema is an example of a bad design:

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

# PITFALLS OF RDBD

- Decomposition
- The obvious solution is that we should decompose this relation.
- As an alternative design, we can use the Decomposition rule:

  If *A* implies *BC* then *A* implies *B* and *A* implies *C*.This gives us the schemas:

  - *branch-customer-schema = (branch-name, branch-city, assets, customer-name)*
  - *customer-loan-schema = (customer-name, loan-number, amount)*

# LOSSLESS JOIN DECOMPOSITION

- Decomposition of R into X and Y is *lossless-join* w.r.t. a set of FDs F if, for every instance *r* that satisfies F:
- (*r*) (*r*) = *r*

- It is always true that *r* (*r*) (*r*)
  - In general, the other direction does not hold! If it does, the decomposition is lossless-join.
- Definition extended to decomposition into 3 or more relations in a straightforward way.
- *It is essential that all decompositions used to deal with redundancy be lossless!*
- *Consider Hourly emps relation.It has attributes SNLRWH and FD R->W causes a violation of 3NF.We dealt this violation by decomposing into SNLRH and RW.*
- *Since R is common to both decomposed relation and*

  *R->W holds,this decomposition is lossles-join*

# LOSSLESS JOIN DECOMPOSITION

- The decomposition of R into X and Y is lossless-join wrt F if and only if the closure of F contains:
  - X ∩ Y → X, or
  - X ∩ Y → Y
- **In** particular,if an fd X->Y holds over relation R and X∩ Y is empty, the decomposition of R into R-Y and XY is lossless.
- Imp observation is repeated decompositions

| A | B | C |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 2 | 8 |
| 1 | 2 | 8 |
| 7 | 2 | 3 |

| A | B | C |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 2 | 8 |

| A | B |
|---|---|
| 1 | 2 |
| 4 | 5 |
| 7 | 2 |

| B | C |
|---|---|
| 2 | 3 |
| 5 | 6 |
| 2 | 8 |

# FUNCTIONAL DEPENDENCIES

- Functional dependency in DBMS, as the name suggests is a relationship between attributes of a table dependent on each other.
- Introduced by E. F. Codd, it helps in preventing data redundancy and gets to know about bad designs.
- To understand the concept thoroughly, let us consider P is a relation with attributes A and B. Functional Dependency is represented by -> (arrow sign)Then the following will represent the functional dependency between attributes with an arrow sign:

A -> B

# FUNCTIONAL DEPENDENCIES

Types of Functional Dependency

Functional Dependency has three forms:

- Trivial Functional Dependency
- Non-Trivial Functional Dependency
- Completely Non-Trivial Functional Dependency

# ARMSTRONG AXIOMS

- Armstrong's Axioms Property of Functional Dependency
- Armstrong's Axioms property was developed by William Armstrong in 1974 to reason about functional dependencies.
- The property suggests rules that hold true if the following are satisfied:
    - Transitivity

      If A->B and B->C, then A->C i.e. a transitive relation.
    - Reflexivity

      A-> B, if B is a subset of A.
    - Augmentation

      The last rule suggests: AC->BC, if A->B

# ARMSTRONG AXIOMS

- If F is a set of functional dependencies then the closure of F, denoted as $F^+$, is the set of all functional dependencies logically implied by F.
- Armstrong's Axioms are a set of rules, that when applied repeatedly, generates a closure of functional dependencies.
  - **Reflexive rule** – If alpha is a set of attributes and beta is_subset_of alpha, then alpha holds beta.
  - **Augmentation rule** – If a → b holds and y is attribute set, then ay → by also holds. That is adding attributes in dependencies, does not change the basic dependencies.
  - Transitivity rule – Same as transitive rule in algebra, if a → b holds and b → c holds, then a → c also holds. a → b is called as a functionally that determines b.

# NORMALIZATION

- Returning to the issue of schema refinement, the first question to ask is whether any refinement is needed!
- If a relation is in a certain *normal form* (BCNF, 3NF etc.), it is known that certain kinds of problems are avoided/minimized.
- This can be used to help us decide whether decomposing the relation will help.
- Role of FDs in detecting redundancy:
  - Consider a relation R with 3 attributes, ABC.
    - No FDs hold: There is no redundancy here.
    - Given A B: Several tuples could have the same A value, and if so, they'll all have the same B value

# 1NF

- 1NF (First Normal Form)
  - a relation R is in 1NF if and only if it has only single-valued attributes (atomic values)
  - EMP_PROJ (SSN, PNO, HOURS, ENAME, PNAME, PLOCATION)
    PLOCATION is not in 1NF (multi-valued attrib.)
  - solution: decompose the relation
    EMP_PROJ2 (SSN, PNO, HOURS, ENAME, PNAME)
    LOC (PNO, PLOCATION)

# 2NF

- 2NF (Second Normal Form)
  - a relation R in 2NF if and only if it is in 1NF and every nonkey column depends on a key not a subset of a key
  - all nonprime attributes of R must be fully functionally dependent on a whole key(s) of the relation, not a part of the key
  - no violation: single-attribute key or no nonprime attribute
- 2NF (Second Normal Form)
  - violation: part of a key $\rightarrow$ nonkey

    EMP_PROJ2 (<u>SSN</u>, <u>PNO</u>, HOURS, ENAME, PNAME)

    SSN $\rightarrow$ ENAME

    PNO $\rightarrow$ PNAME
  - solution: decompose the relation

    EMP_PROJ3 (<u>SSN</u>, <u>PNO</u>, HOURS)

    EMP (<u>SSN</u>, ENAME)

    PROJ (<u>PNO</u>, PNAME)

# 3NF

- a relation R in 3NF if and only if it is in 2NF and every nonkey column does not depend on another nonkey column
- all nonprime attributes of R must be non-transitively functionally dependent on a key of the relation

- violation: nonkey → nonkey

  - SUPPLIER (SNAME, STREET, CITY, STATE, TAX)

    SNAME → STREET, CITY, STATE

    STATE → TAX (nonkey → nonkey)

    SNAME → STATE → TAX (transitive FD)

  - solution: decompose the relation

    SUPPLIER2 (SNAME, STREET, CITY, STATE)

    TAXINFO (STATE, TAX)

# MVDs

- Suppose that we have a relation with attributes *course*, *teacher*, and *book*, which we denote as *CTB*.
- The meaning of a tuple is that teacher *T* can teach course *C*, and book *B* is a recommended text for the course.
- There are no FDs; the key is *CTB*. However, the recommended texts for a course are independent of the instructor

| *course* | *teacher* | *book* |
|---|---|---|
| Physics101 | Green | Mechanics |
| Physics101 | Green | Optics |
| Physics101 | Brown | Mechanics |
| Physics101 | Brown | Optics |
| Math301 | Green | Mechanics |
| Math301 | Green | Vectors |
| Math301 | Green | Geometry |

# MVDs

There are three points to note here:

- The relation schema *CTB* is in BCNF; thus we would not consider decomposing it further if we looked only at the FDs that hold over *CTB*.
- There is redundancy. The fact that Green can teach Physics101 is recorded once per recommended text for the course. Similarly, the fact that Optics is a text for Physics101 is recorded once per potential teacher.
- The redundancy can be eliminated by decomposing *CTB* into *CT* and *CB*.
- Let *R* be a relation schema and let *X* and *Y* be subsets of the attributes of *R*. Intuitively,
- the **multivalued dependency** *X !! Y* is said to hold over *R* if, in every legal

# MVDs

- The redundancy in this example is due to the constraint that the texts for a course are independent of the instructors, which cannot be expressed in terms of FDs.
- This constraint is an example of a *multivalued dependency*, or MVD
- Ideally, we should model this situation using two binary relationship sets, Instructors with attributes *CT* and Text with attributes *CB*.
- Because these are two essentially independent relationships, modeling them with a single ternary relationship set with attributes *CTB* is inappropriate.

# MVDs

- Three of the additional rules involve only MVDs:
  - MVD Complementation: If $X \rightarrow\rightarrow Y$, then $X \rightarrow\rightarrow R - XY$
  - MVD Augmentation: If $X \rightarrow\rightarrow Y$ and $W > Z$, then $WX \rightarrow\rightarrow YZ$.
  - MVD Transitivity: If $X \rightarrow\rightarrow Y$ and $Y \rightarrow\rightarrow Z$, then $X \rightarrow\rightarrow (Z - Y)$.
- **Fourth Normal Form**
- *R* is said to be in **fourth normal form (4NF)** if for every MVD $X \rightarrow\rightarrow Y$ that holds over *R*, one of the following statements is true:
  - *Y subset of X* or $XY = R$, or
  - *X* is a superkey.

# JDs

- A join dependency is a further generalization of MVDs.
- A **join dependency** (JD) ∞{ *R*1,….. *Rn }* is said to hold over a relation *R* if *R*1,…. *Rn* is a lossless-join decomposition of *R*.
- An MVD *X* ->-> *Y* over a relation *R* can be expressed as the join dependency ∞ { XY,X(R−Y)*}*
- As an example, in the *CTB* relation, the MVD *C* ->->*T* can be expressed as the join dependency ∞{ CT, CB*}*
- Unlike FDs and MVDs, there is no set of sound and complete inference rules for JDs.

# 4NF

- Fourth normal form,
- It should meet all the requirement of 3NF
- Attribute of one or more rows in the table should not result in more than one rows of the same table leading to multi-valued dependencies
- For a table to satisfy the Fourth Normal Form, it should satisfy the following two conditions:
  - It should be in the **Boyce-Codd Normal Form**.
  - And, the table should not have any **Multi-valued Dependency**.

# 5NF

- A relation schema *R* is said to be in **fifth normal form (5NF)** if for every JD ∞{ *R*1,…. *Rn* } that holds over *R*, one of the following statements is true:
  - *Ri* = *R* for some *i*, or
  - The JD is implied by the set of those FDs over *R* in which the left side is a key for *R*.

- A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.
- 5NF is satisfied when all the tables are broken into as many tables as possible in order to avoid redundancy.
- 5NF is also known as Project-join normal form (PJ/NF).
- If a relation schema is in 3NF and each of its keys consists of a single attribute,it is also in 5NF.

# MODULE IV
# TRANSACTION MANAGEMENT

| CLOs | Course Learning Outcome |
|---|---|
| CLO 16 | Discuss the concept of transaction, transaction state. |
| CLO 17 | Understand atomicity and durability, concurrent executions. |
| CLO 18 | Summarize the concept of serializability, recoverability. |
| CLO 19 | Discuss the concurrency control and various protocols. |
| CLO20 | Understand the concept of multi version schemes, deadlock handling and recovery. |

# TRANSACTION CONCEPT

- A Transaction is a *unit* of program execution that accesses and possibly updates various data items.

- Example transaction to transfer $50 from account A to account B:
    1. read(*A*)
    2. *A* := *A* – 50
    3. write(*A*)
    4. read(*B*)
    5. *B* := *B* + 50
    6. write(*B)*
- Two main issues to deal with:
- Failures of various kinds, such as hardware failures and system crashes Concurrent execution of multiple transactions

# TRANSACTION CONCEPT

- Durability requirement —
- once the user has been notified that the transaction has completed (i.e., the transfer of the $50 has taken place),
- the updates to the database by the transaction must persist even if there are software or hardware failures.
- Example of Fund Transfer Transaction to transfer $50 from account A to account B:

1. read(*A*)
2. *A* := *A* – 50
3. write(*A*)
4. read(*B*)
5. *B* := *B* + 50
6. write(*B*)

# TRANSACTION CONCEPT

- Example of Fund Transfer Isolation requirement — if between steps 3 and 6,
- another transaction T2 is allowed to access the partially updated database, it will see an inconsistent database .
- T1 T2

1. read(*A*)
2. *A* := *A* – 50
3. write(*A*)

read(A), read(B), print(A+B)

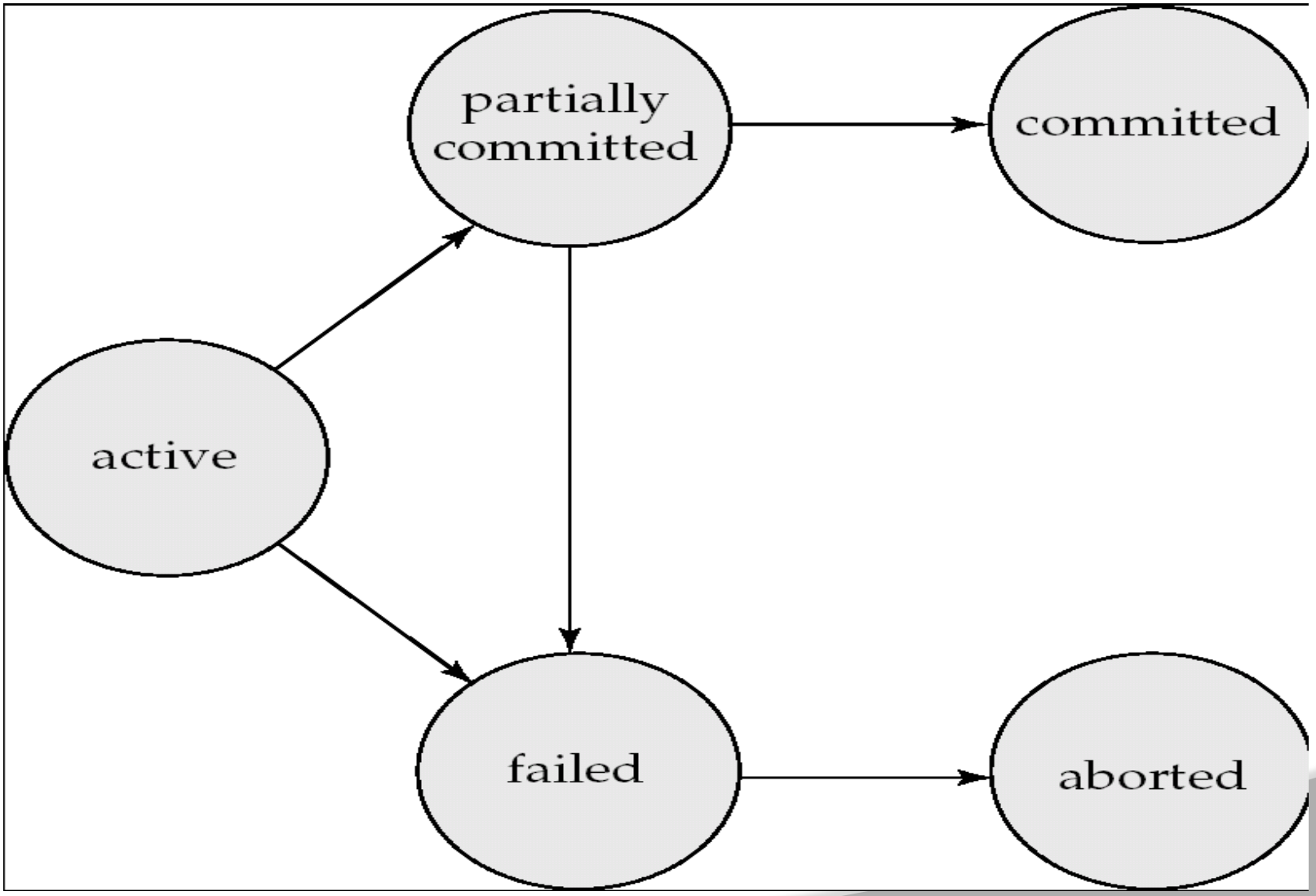
4. read(*B*)
5. *B* := *B* + 50
6. write(*B* )

- Isolation can be ensured trivially by running transactions serially that is, one after the other. However, executing multiple transactions concurrently has significant benefits.

# TRANSACTION STATE

- Active – the initial state; the transaction stays in this state while it is executing
- Partially committed – after the final statement has been executed.
- Failed -- after the discovery that normal execution can no longer proceed.
- Aborted – after the transaction has been rolled back and the database restored to its State prior to the start of the transaction. Two options after it has been aborted: restart the transaction can be done only if no internal logical error kill the transaction
- Committed – after successful completion.

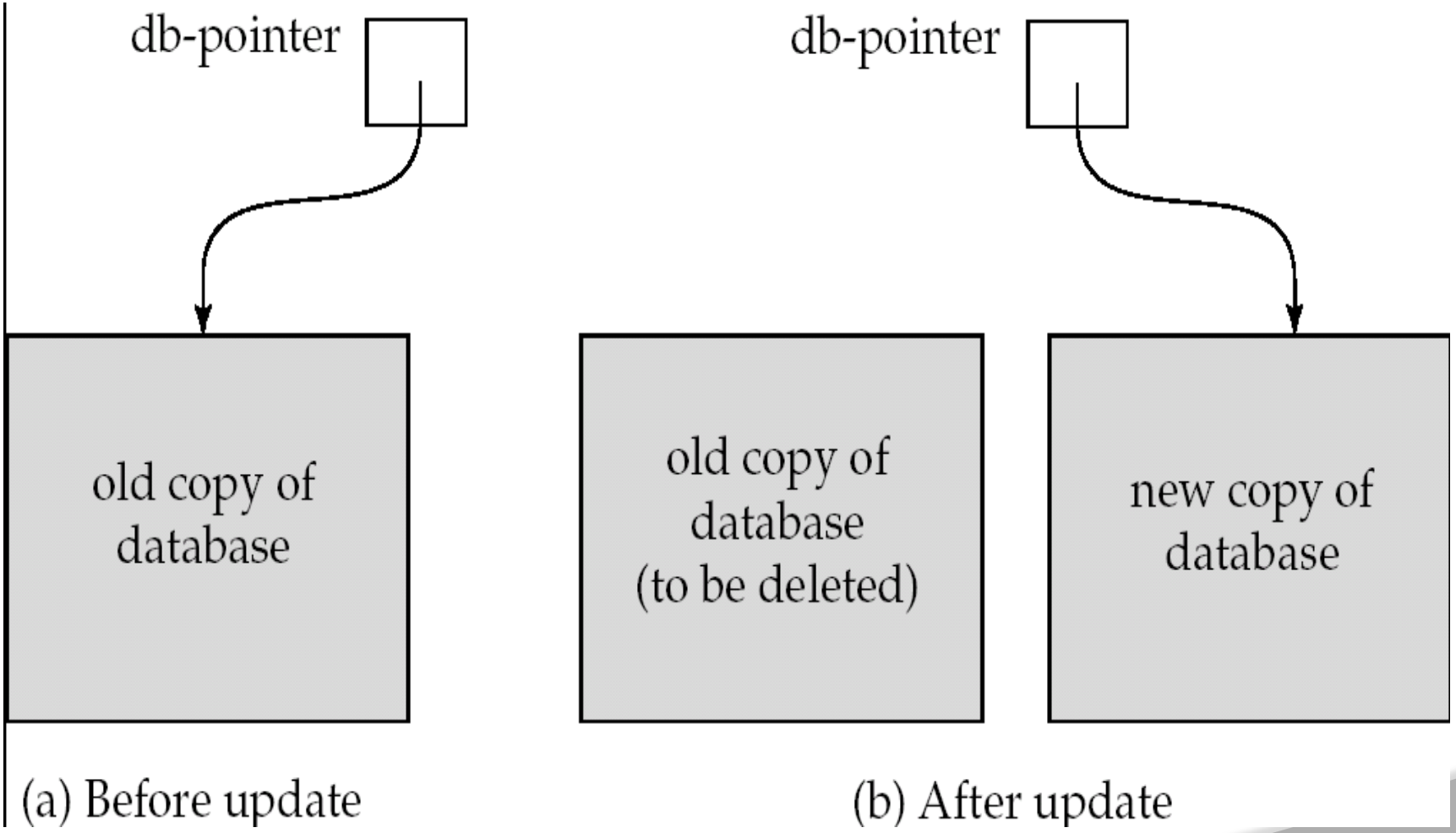
# TRANSACTION STATE



# IMPLEMENTATION OF ATOMICITY AND DURABILITY

- The recovery-management component of a database system implements the support for atomicity and durability.
- Example of the *shadow-database* scheme:all updates are made on a *shadow copy* of the database .
- db_pointer is made to point to the updated shadow copy after the transaction reaches partial commit and all updated pages have been flushed to disk.
- db_pointer always points to the current consistent copy of the database.In case transaction fails, old consistent copy pointed to by db_pointer can be used, and the shadow copy can be deleted.

# IMPLEMENTATION OF ATOMICITY AND DURABILITY



# CONCURRENT EXECUTIONS

- Multiple transactions are allowed to run concurrently in the system. Advantages are: increased processor and disk utilization, leading to better transaction throughput.
- Example one transaction can be using the CPU while another is reading from or writing to the disk reduced average response time for transactions:
- short transactions need not wait behind long ones Concurrency control schemes – mechanisms to achieve isolation that is, to control the interaction among the concurrent transactions in order to prevent them from destroying the consistency of the database.
- Schedule – a sequences of instructions that specify the chronological order in which instructions of concurrent transactions are executed a schedule for a set of transactions must consist of all instructions of those transactions must preserve the order in which the instructions appear in each individual transaction.

# CONCURRENT EXECUTIONS

- A transaction that successfully completes its execution will have commit instructions as the last statement by default transaction assumed to execute commit instruction as its last step
- A transaction that fails to successfully complete its execution will have an abort instruction as the last statement.
- Schedule 1
- Let $T_1$ transfer $50 from *A* to *B*, and $T_2$ transfer 10% of the balance from *A* to *B.*
- A serial schedule in which $T_1$ is followed by $T_2$ :

| $T_1$ | $T_2$ |
|---|---|
| read($A$) | |
| $A := A - 50$ | |
| write ($A$) | |
| read($B$) | |
| $B := B + 50$ | |
| write($B$) | |
| | read($A$) |
| | $temp := A * 0.1$ |
| | $A := A - temp$ |
| | write($A$) |
| | read($B$) |
| | $B := B + temp$ |
| | write($B$) |

| $T_1$ | $T_2$ |
|---|---|
| | read($A$) |
| | $temp := A * 0.1$ |
| | $A := A - temp$ |
| | write($A$) |
| | read($B$) |
| | $B := B + temp$ |
| | write($B$) |
| read($A$) | |
| $A := A - 50$ | |
| write($A$) | |
| read($B$) | |
| $B := B + 50$ | |
| write($B$) | |

# SERIALIZABILITY

- Basic Assumption – Each transaction preserves database consistency.
- Thus serial execution of a set of transactions preserves database consistency.A (possibly concurrent) schedule is serializable if it is equivalent to a serial schedule. Different forms of schedule equivalence give rise to the notions of:

ility

| $T_1$ | $T_2$ |
|---|---|
| read($A$) | |
| write($A$) | |
| | read($A$) |
| | write($A$) |
| read($B$) | |
| write($B$) | |
| | read($B$) |
| | write($B$) |

| $T_1$ | $T_2$ |
|---|---|
| read($A$) | |
| write($A$) | |
| read($B$) | |
| write($B$) | |
| | read($A$) |
| | write($A$) |
| | read($B$) |
| | write($B$) |

| $T_3$ | $T_4$ |
|---|---|
| read($Q$) | |
| | write($Q$) |
| write($Q$) | |

| $T_3$ | $T_4$ | $T_6$ |
|---|---|---|
| read($Q$) | | |
| | write($Q$) | |
| write($Q$) | | |
| | | write($Q$) |

# SERIALIZABILITY

- A schedule *S* is view serializable if it is view equivalent to a serial schedule. Every conflict serializable schedule is also view serializable.
- Below is a schedule which is view-serializable but *not* conflict serializable.
- What serial schedule is above equivalent to?
- Every view serializable schedule that is not conflict serializable has blind writes.

Other Not

| $T_1$ | $T_5$ |
|---|---|
| read($A$) | |
| $A := A - 50$ | |
| write($A$) | |
| | read($B$) |
| | $B := B - 10$ |
| | write($B$) |
| read($B$) | |
| $B := B + 50$ | |
| write($B$) | |
| | read($A$) |
| | $A := A + 10$ |
| | write($A$) |

# RECOVERABILITY

- Recoverable schedule — if a transaction $T_j$ reads a data item previously written by a transaction $T_i$, then the commit operation of $T_i$ appears before the commit operation of $T_j$. The following schedule (Schedule 11) is not recoverable if $T_9$ commits immediately after the read

| $T_8$ | $T_9$ |
|---|---|
| read($A$) | |
| write($A$) | |
| | read($A$) |
| read($B$) | |

- If $T_8$ should abort, $T_9$ would have read (and possibly shown to the user) an inconsistent database state. Hence, database must ensure that schedules are recoverable.
- Cascading Rollbacks

  Cascading rollback – a single transaction failure leads to a series of transaction rollbacks. Consider the following schedule where none of the transactions has yet committed

# LOCK-BASED PROTOCOLS

- A lock is a mechanism to control concurrent access to a data item

| | S | X |
|---|---|---|
| S | true | false |
| X | false | false |

- Fig:Lock-compatibility matrix

- Data items can be locked in two modes :

1. *exclusive (X) mode*. Data item can be both read as well as written. X-lock is requested using lock-X instruction.
2. *shared (S) mode*. Data item can only be read. S-lock is requested using lock-S instruction.

# LOCK-BASED PROTOCOLS

- Example :if a transaction performing locking:

  $T_2$: lock-S*(A)*;
  read *(A)*;
  unlock*(A)*;
  lock-S*(B)*;
  read *(B)*;
  unlock*(B)*;
  display*(A+B)*

- Locking as above is not sufficient to guarantee serializability — if *A* and *B* get updated in-between the read of *A* and *B*, the displayed sum would be wrong.

# LOCK-BASED PROTOCOLS

- A locking protocol is a set of rules followed by all transactions while requesting and releasing locks.
- Locking protocols restrict the set of possible schedules. Pitfalls of Lock-Based Protocols Consider the partial schedule Neither $T_3$ nor $T_4$ can make progress — executing lock-S*(B)* causes $T_4$ to wait for $T_3$ to release its lock on *B*, while executing lock-X*(A)* causes $T_3$ to wait for $T_4$ to release its lock on *A*.
- Such a situation is called a deadlock. To handle a deadlock one of $T_3$ or $T_4$ must be rolled back and its locks released.
- The potential for deadlock exists in most locking protocols. Deadlocks are a necessary evil.

# TIMESTAMP -BASED PROTOCOLS

- Each transaction is issued a timestamp when it enters the system. If an old transaction Ti has time-stamp TS(Ti), a new transaction Tj is assigned time-stamp TS(Tj) such that TS(Ti) <TS(Tj).
- The protocol manages concurrent execution such that the time-stamps determine the serializability order.In order to assure such behavior, the protocol maintains for each data Q two timestamp values:
- W-timestamp(Q) is the largest time-stamp of any transaction that executed write(Q) successfully.
- R-timestamp(Q) is the largest time-stamp of any transaction that executed read(Q) successfully.
- The timestamp ordering protocol ensures that any conflicting read and write operations are executed in timestamp order.

# VALIDATION -BASED PROTOCOLS

- Execution of transaction Ti is done in three phases.
  1. Read and execution phase: Transaction Ti writes only to temporary local variables
  2. Validation phase: Transaction Ti performs a ``validation test'' to determine if local variables can be written without violating serializability.
  3. Write phase: If Ti is validated, the updates are applied to the database; otherwise, Ti is rolled back.

# VALIDATION -BASED PROTOCOLS

- The three phases of concurrently executing transactions can be interleaved, but each transaction must go through the three phases in that order.
- Assume for simplicity that the validation and write phase occur together, atomically and serially i.e., only one transaction executes validation/write at a time. Also called as optimistic concurrency control since transaction executes fully in the hope that all will go well during validation.
- Each transaction Ti has 3 timestamps
  - Start(Ti) : the time when Ti started its execution
  - Validation(Ti): the time when Ti entered its validation phase
  - Finish(Ti) : the time when Ti finished its write phase Serializability order is determined by timestamp given at validation time, to increase concurrency.

# VALIDATION -BASED PROTOCOLS

- Example of schedule produced using validation

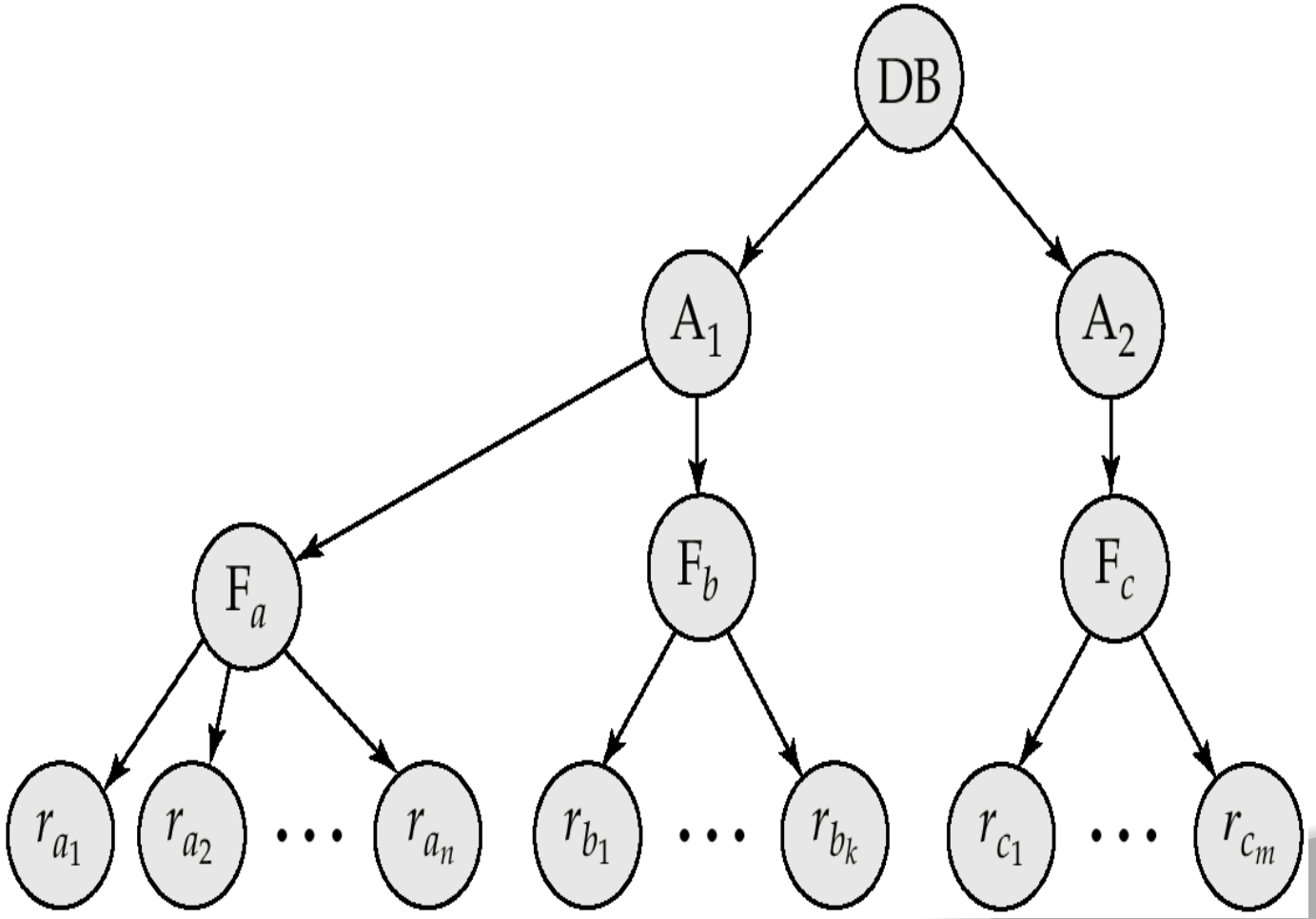
| $T_{14}$ | $T_{15}$ |
|---|---|
| read ($B$) | |
| | read ($B$)<br>$B := B-50$<br>read ($A$)<br>$A := A+50$ |
| read ($A$)<br>(*validate*)<br>display ($A+B$) | |
| | (*validate*)<br>write ($B$)<br>write ($A$) |

# MULTIPLE GRANULARITIES

- Allow  data items to be of various sizes and define a hierarchy of data granularities,
- where the small granularities are nested within larger ones Can be represented graphically as a tree
- When a transaction locks a node in the tree explicitly, it implicitly locks all the node's descendents in the same mode.
- Granularity of locking (level in tree where locking is done):ine granularity (lower in tree):
- high concurrency, high locking overhead coarse granularity (higher in tree): low locking overhead, low concurrency

# MULTIPLE GRANULARITIES

Example of Granularity Hierarchy



## Example of Granularity Hierarchy

- The compatibility matrix for all lock modes is:

| | IS | IX | S | S IX | X |
|---|---|---|---|---|---|
| IS | ✓ | ✓ | ✓ | ✓ | × |
| IX | ✓ | ✓ | × | × | × |
| S | ✓ | × | ✓ | × | × |
| S IX | ✓ | × | × | × | × |
| X | × | × | × | × | × |

# MULTIVERSION SCHEMES

- Multiversion concurrency control techniques keep the old values of a data item when the item is updated.
- Several versions (values) of an item are maintained. When a transaction requires access to an item, an appropriate version is chosen to maintain the serialisability of the concurrently executing schedule, if possible.
- The idea is that some read operations that would be rejected in other techniques can still be accepted, by reading an older version of the item to maintain serialisability.
- An obvious drawback of multiversion techniques is that more storage is needed to maintain multiple versions of the database items. However, older versions may have to be maintained anyway
- For example, for recovery purpose. In addition, some database applications require older versions to be kept to maintain a history of the evolution of data item values.

# MULTIVERSION SCHEMES

- The extreme case is a temporal database, which keeps track of all changes and the items at which they occurred.
- In such cases, there is no additional penalty for multiversion techniques, since older versions are already maintained
- In this multiversion two-phase locking scheme, reads can proceed concurrently with a write operation an arrangement not permitted under the standard two-phase locking schemes.
- The cost is that a transaction may have to delay its commit until it obtains exclusive certify locks on all items it has updated.
- It can be shown that this scheme avoids cascading aborts, since transactions are only allowed to read the version X that was written by committed transaction. However, deadlock may occur

# DEADLOCK

- Another problem that may be introduced by 2PL protocol is deadlock. The formal definition of deadlock will be discussed below.
- Example is used to give you an intuitive idea about the deadlock situation.
- The two transactions that follow the 2PL protocol can be interleaved as shown here:
- At time step 5, it is not possible for T1â€™ to acquire an exclusive lock on X as there is already a shared lock on X held by T2â€™.
- Therefore, T1â€™ has to wait. Transaction T2â€™ at time step 6 tries to get an exclusive lock on Y, but it is unable to as T1â€™ has a shared lock on Y already. T2â€™ is put in waiting too.
- Therefore, both transactions wait fruitlessly for the other to release a lock. This situation is known as a deadly embrace or deadlock. The above schedule would terminate in a deadlock.

# DEADLOCK

| Time | $T_1'$ | $T_2'$ |
|---|---|---|
| 1 | read_lock(Y); | |
| 2 | read_item(Y); | |
| 3 | | read_lock(X); |
| 4 | | read-item(X); |
| 5 | write_lock(X); | |
| | wait | |
| 6 | | write_lock(Y); |
| | | wait |
| | ... | ... |

# RECOVERY: FAILURE CLASSIFICATION

- A transaction has to abort when it fails to execute or when it reaches a point from where it can’t go any further.
- This is called transaction failure where only a few transactions or processes are hurt.
- Reasons for a transaction failure could be −
- Logical errors − Where a transaction cannot complete because it has some code error or any internal error condition.
- System errors − Where the database system itself terminates an active transaction because the DBMS is not able to execute it, or it has to stop because of some system condition.
- For example, in case of deadlock or resource unavailability, the system aborts an active transaction.

# RECOVERY: FAILURE CLASSIFICATION

- There are problems − external to the system − that may cause the system to stop abruptly and cause the system to crash.
- For example, interruptions in power supply may cause the failure of underlying hardware or software failure.
- Examples may include operating system errors.
- Disk Failure
- In early days of technology evolution, it was a common problem where hard-disk drives or storage drives used to fail frequently.
- Disk failures include formation of bad sectors, unreachability to the disk, disk head crash or any other failure, which destroys all or a part of disk storage.

# STORAGE STRUCTURE

- We have already described the storage system. In brief, the storage structure can be divided into two categories −
- Volatile storage − As the name suggests, a volatile storage cannot survive system crashes.
- Volatile storage devices are placed very close to the CPU; normally they are embedded onto the chipset itself.
- For example, main memory and cache memory are examples of volatile storage. They are fast but can store only a small amount of information.
- Non-volatile storage − These memories are made to survive system crashes.
- They are huge in data storage capacity, but slower in accessibility. Examples may include hard-disks, magnetic tapes, flash memory, and non-volatile (battery backed up) RAM.

# RECOVERY AND ATOMICITY

- When a system crashes, it may have several transactions being executed and various files opened for them to modify the data items.
- Transactions are made of various operations, which are atomic in nature. But according to ACID properties of DBMS, atomicity of transactions as a whole must be maintained, that is, either all the operations are executed or none.
- When a DBMS recovers from a crash, it should maintain the following −
  - It should check the states of all the transactions, which were being executed.

# RECOVERY AND ATOMICITY

- A transaction may be in the middle of some operation; the DBMS must ensure the atomicity of the transaction in this case.
- It should check whether the transaction can be completed now or it needs to be rolled back.
- No transactions would be allowed to leave the DBMS in an inconsistent state.
- There are two types of techniques, which can help a DBMS in recovering as well as maintaining the atomicity of a transaction –
- Maintaining the logs of each transaction, and writing them onto some stable storage before actually modifying the database.
- Maintaining shadow paging, where the changes are done on a volatile memory, and later, the actual database is updated.

# LOG BASED RECOVERY

- Log is a sequence of records, which maintains the records of actions performed by a transaction.
- It is important that the logs are written prior to the actual modification and stored on a stable storage media, which is failsafe.
- Log-based recovery works as follows −
- The log file is kept on a stable storage media.
- When a transaction enters the system and starts execution, it writes a log about it.

# LOG BASED RECOVERY

<$T_n$, Start>
When the transaction modifies an item X, it write logs as follows −
<$T_n$, X, $V_1$, $V_2$>
It reads $T_n$ has changed the value of X, from $V_1$ to $V_2$.
When the transaction finishes, it logs −
<$T_n$, commit>

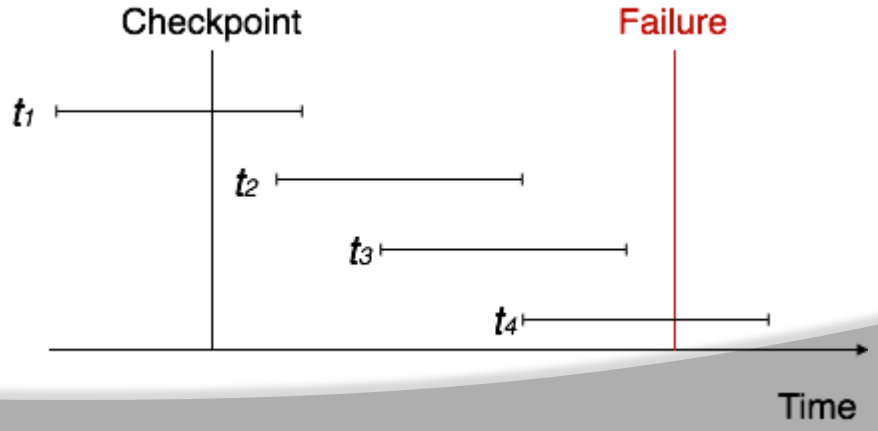
- The database can be modified using two approaches −
- Deferred database modification − All logs are written on to the stable storage and the database is updated when a transaction commits.
- Immediate database modification − Each log follows an actual database modification. That is, the database is modified immediately after every operation

# RECOVERY WITH CONCURRENT TRANSACTIONS

- When more than one transaction are being executed in parallel, the logs are interleaved.
- At the time of recovery, it would become hard for the recovery system to backtrack all logs, and then start recovering. To ease this situation, most modern DBMS use the concept of 'checkpoints'.
- Checkpoint
- Keeping and maintaining logs in real time and in real environment may fill out all the memory space available in the system. As time passes, the log file may grow too big to be handled at all.
- Checkpoint is a mechanism where all the previous logs are removed from the system and stored permanently in a storage disk.
- Checkpoint declares a point before which the DBMS was in consistent state, and all the transactions were committed.
- Recovery
- When a system with concurrent transactions crashes and recovers, it behaves in the following manner

# RECOVERY WITH CONCURRENT TRANSACTIONS

- The recovery system reads the logs backwards from the end to the last checkpoint.
- It maintains two lists, an undo-list and a redo-list.
- If the recovery system sees a log with <Tn, Start> and <Tn, Commit> or just <Tn, Commit>, it puts the transaction in the redo-list.
- If the recovery system sees a log with <Tn, Start> but no commit or abort log found, it puts the transaction in undo-list.
- All the transactions in the undo-list are then undone and their logs are removed. All the transactions in the redo-list and their previous logs are removed and then redone before saving their logs.



# MODULE V
# DATA STORAGE AND QUERY PROCESSING

| CLOs | Course Learning Outcome |
|---|---|
| CLO 21 | Knowledge about the physical storage media, magnetic disks, storage access. |
| CLO 22 | Apply working with file organization, organization of records in files. |
| CLO 23 | Understand ordered indices, b+-tree index files, b-tree index files, static hashing, dynamic hashing. |
| CLO 24 | Comparison of ordered indexing and hashing. |
| CLO 25 | Illustrate query processing: overview, measures of query cost. |

| CLOs | Course Learning Outcome |
|---|---|
| CLO 25 | Learning method of separation of variables |
| CLO 26 | Solving the heat equation and wave equation in subject to boundary conditions |
| CLO 27 | Understand the concept of partial differential equations to the real-world problems of electromagnetic and fluid dynamics |

# OVERVIEW OF PHYSICAL STORAGE MEDIA

- Storage media are classified by speed of access, cost per unit of data to buy the media, and by the medium's reliability.
- Unfortunately, as speed and cost go up, the reliability does down.
- Cache is the fastest and the most costly for of storage.
- The type of cache referred to here is the type that is typically built into the CPU chip and is 256KB, 512KB, or 1MB.
- Thus, cache is used by the operating system and has no application to database, per se.

# OVERVIEW OF PHYSICAL STORAGE MEDIA

- Main memory is the volatile memory in the computer system that is used to hold programs and data.
- While prices have been dropping at a staggering rate, the increases in the demand for memory have been increasing faster.
- Today's 32-bit computers have a limitation of 4GB of memory. This may not be sufficient to hold the entire database and all the associated programs, but the more memory available will increase the response time of the DBMS.
- There are attempts underway to create a system with the most memory that is cost effective, and to reduce the functionality of the operating system so that only the DBMS is supported.
- so that system response can be increased. However, the contents of main memory are lost if a power failure or system crash occurs.

# OVERVIEW OF PHYSICAL STORAGE MEDIA

- Flash memory is also referred to as *electrically erasable programmable read-only memory (EEPROM)*. Since it is small (5 to 10MB) and expensive, it has little or no application to the DBMS.
- Magnetic-disk storage is the primary medium for long-term on-line storage today. Prices have been dropping significantly with a corresponding increase in capacity.
- New disks today are in excess of 20GB. Unfortunately, the demands have been increasing and the volume of data has been increasing faster.
- The organizations using a DBMS are always trying to keep up with the demand for storage. This media is the most cost-effective for on-line storage for large databases.
- Optical storage is very popular, especially CD-ROM systems. This is limited to data that is read-only. It can be reproduced at a very low-cost and it is expected to grow in popularity, especially for replacing written manuals.

# OVERVIEW OF PHYSICAL STORAGE MEDIA

- Tape storage is used for backup and archival data. It is cheaper and slower than all of the other forms.
- but it does have the feature that there is no limit on the amount of data that can be stored, since more tapes can be purchased.
- As the tapes get increased capacity, however, restoration of data takes longer and longer, especially when only a small amount of data is to be restored.
- This is because the retrieval is sequential, the slowest possible method

# MAGNETIC DISKS

- A typical large commercial database may require hundreds of disks!

**Physical Characteristics of Disks**

- Disks are actually relatively simple. There is normally a collection of platters on a spindle.
- Each platter is coated with a magnetic material on both sides and the data is stored on the surfaces.
- There is a read-write head for each surface that is on an arm assembly that moves back and forth.
- A motor spins the platters at a high constant speed, The surface is divided into a set of tracks (circles).
- These tracks are divided into a set of sectors, which is the smallest unit of data that can be written or read at one time.
- Sectors can range in size from 31 bytes to 4096 bytes, with 512 bytes being the most common.
- A collection of a specific track from both surfaces and from all of the platters is called a cylinder.

# MAGNETIC DISKS

**Performance Measures of Disks**

- ***Seek time*** is the time to reposition the head and increases with the distance that the head must move. Seek times can range from 2 to 30 milliseconds. *Average seek time* is the average of all seek times and is normally one-third of the worst-case seek time.
- ***Rotational latency time*** is the time from when the head is over the correct track until the data rotates around and is under the head and can be read. When the rotation is 120 rotations per second, the rotation time is 8.35 milliseconds. Normally, the *average rotational latency time* is one-half of the rotation time.
- ***Access time*** is the time from when a read or write request is issued to when the data transfer begins. It is the sum of the seek time and latency time.
- ***Data-transfer rate*** is the rate at which data can be retrieved from the disk and sent to the controller. This will be measured as megabytes per second.

# MAGNETIC DISKS

RAIDs are Redundant Arrays of Inexpensive Disks. There are six levels of organizing these disks:

0 -- Non-redundant Striping

1 -- Mirrored Disks

2 -- Memory Style Error Correcting Codes

3 -- Bit Interleaved Parity

4 -- Block Interleaved Parity

5 -- Block Interleaved Distributed Parity

6 -- P + Q Redundancy

Tertiary Storage

This is commonly optical disks and magnetic tapes

# STORAGE ACCESS

- Programs in a DBMS make requests (that is, calls) on the buffer manager when they need a block from a disk.
- If the block is already in the buffer, the requester is passed the address of the block in main memory.
- If the block in not in the buffer, the buffer manager first allocates space in the buffer for the block, through out some other block, if required, to make space for the new block.
- If the block that is to be thrown out has been modified, it must first be written back to the disk.
- The internal actions of the buffer manager are transparent to the programs that issue disk-block requests.

# STORAGE ACCESS

- ***Replacement strategy.*** When there is no room left in the buffer, a block must be removed from the buffer before a new one can be read in.
- Typically, operating systems use a least recently use (LRU) scheme. There is also a Most Recent Used (MRU) that can be more optimal for DBMSs.
- ***Pinned blocks.*** A block that is not allowed to be written back to disk is said to be pinned. This could be used to store data that has not been committed yet. *Forced output of blocks*.
- There are situations in which it is necessary to write back to the block to the disk, even though the buffer space is not currently needed. This might be done during system lulls, so that when activity picks up, a write of a modified block can be avoided in peak periods.

# FILE ORGANIZATION

- Fixed-Length Records
- Suppose we have a table that has the following organization:
- type deposit = record
  branch-name : char(22);
  account-number : char(10);
  balance : real;
  end
- If each character occupies 1 byte and a real occupies 8 bytes, then this record occupies 40 bytes.
- If the first record occupies the first 40 bytes and the second record occupies the second 40 bytes, etc. we have some problems.
- It is difficult to delete a record, because there is no way to indicate that the record is deleted
- Unless the block size happens to be a multiple of 40 (which is extremely unlikely), some records will cross block boundaries. It would require two block access to read or write such a record.

# FILE ORGANIZATION

- Variable-Length Records
- We can use variable length records:
  - Storage of multiple record types in one file.
  - Record types that allow variable lengths for one or more fields
  - Record types that allow repeating fields.
- A simple method for implementing variable-length records is to attach a special *end-of-record* symbol at the end of each record. But this has problems:
- To easy to reuse space occupied formerly by a deleted record.
- There is no space in general for records to grow. If a variable-length record is updated and needs more space, it must be moved. This can be very costly.
- It could be solved by making a variable-length into a fixed length.By using pointers to point to fixed length records, chained together by pointers.

# ORGANIZATION OF RECORDS IN FILES

- Heap File Organization
  - Any record can be placed anywhere in the file.
  - There is no ordering of records and there is a single file for each relation.
- Sequential File Organization
  - Records are stored in sequential order based on the primary key.
- Hashing File Organization
  - Any record can be placed anywhere in the file. A hashing function is computed on some attribute of each record.
  - The function specifies in which block the record should be placed.
- Clustering File Organization
  - Several different relations can be stored in the same file.
  - Related records of the different relations can be stored in the same block.

# ORGANIZATION OF RECORDS IN FILES

- A RDBMS needs to maintain data about the relations, such as the schema. This is stored in a data dictionary.
- Names of the relations
- Names of the attributes of each relation
- Domains and lengths of attributes
- Names of views, defined on the database, and definitions of those views
- Integrity constraints
- Names of authorized users
- Accounting information about users
- Number of tuples in each relation
- Method of storage for each relation (clustered/non-clustered)
- Name of the index
- Name of the relation being indexed
- Attributes on which the index in defined
- Type of index formed

# INDEXING AND HASHING: BASIC CONCEPTS

- An index for a file is like a catalog for a book in the library. Cards in the catalog are stored in order with portions of the catalog order by author's name, book title, or subject.
- Items in the database are catalogued with indices based on keys. When a table is defined, it has a primary key; however, it can have additional keys defined.
- Typical databases are too large to search sequentially looking for specific records and more sophisticated indexing techniques are employed. The two basic kinds of indices are:

# ORDERED INDICES

- Dense and Sparse Indices
- This can be improved by adding another data structure (especially for random access).
- Dense Index. An *index record* (or *index entry*) appears for every search-key value, containing the value and the location for the first data record with that valueSparse Index.
- An index record is created only for some of the values, which is the only difference between the two versions in terms of the data and its structure.
- With the sparse index, the system has to locate the largest value in the index that does not exceed the search key. From that point the records must be checked sequentially until a match is found.

# ORDERED INDICES

**Ordered indices**

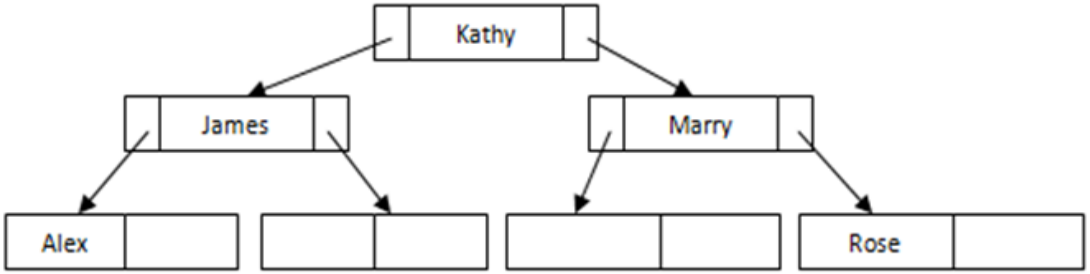
- Hash indices
- Different techniques are evaluated on the basis of several opposing factors:
    - Access types: The access could be for a specific value or a specified range.
    - Access time: The amount of time it takes to find a particular data item or set of items.
    - Insertion time: The time it takes to insert a new data item. This value includes the time it takes to find the correct place to insert the new data items, as well as update the index structure.

# B+-TREE INDEX FILES

- B+ tree is used to store the records in the secondary memory. If the records are stored using this concept, then those files are called as B+ tree index files.
- Since this tree is balanced and sorted, all the nodes will be at same distance and only leaf node has the actual value, makes searching for any record easy and quick in B+ tree index files.
- Even insertion/deletion in B+ tree does not take much time. Hence B+ tree forms an efficient method to store the records.
- Searching, inserting and deleting a record is done in the same way we have seen above. Since it is a balance tree, it searches for the position of the records in the file, and then it fetches/inserts /deletes the records.
- In case it finds that tree will be unbalanced because of insert/delete/update, it does the proper re-arrangement of nodes so that definition of B+ tree is not changed.

# B-TREE INDEX FILES

- B tree index file is similar to B+ tree index files, but it uses binary search concepts.
- In this method, each root will branch to only two nodes and each intermediary node will also have the data. And leaf node will have lowest level of data.
- However, in this method also, records will be sorted. Since all intermediary nodes also have records, it reduces the traversing till leaf node for the data. A simple B tree can be represented as below:



# STATIC HASHING

- In a hash file organization, we obtain the address of the disk block (actually a *bucket* that can contain one or more records) containing the desired record by computing a function on the search-key value of the record.
- If **K** is the set of all search-key values, **B** is the set of all bucket addresses, *h* is the hash function, we can computer the address of the bucket to insert a record with the search-key $\mathbf{K_i}$ using $h(\mathbf{K_i})$.
- Assuming there is space in the bucket, we can simply insert the record. We locate the record with the search-key $\mathbf{K_i}$ using $h(\mathbf{K_i})$. Deletion is done the same way.
- However if it turns out the two records have the same hash value, $h(\mathbf{K_5}) = h(\mathbf{K_7})$, then we do a sequence search on the bucket for the record that is desired.

# STATIC HASHING

- In static hashing, when a search-key value is provided, the hash function always computes the same address.
- For example, if mod-4 hash function is used, then it shall generate only 5 values. The output address shall always be same for that function.
- The number of buckets provided remains unchanged at all times. Insertion − When a record is required to be entered using static hash, the hash function h computes the bucket address for search key K, where the record will be stored.
- Bucket address = h(K)

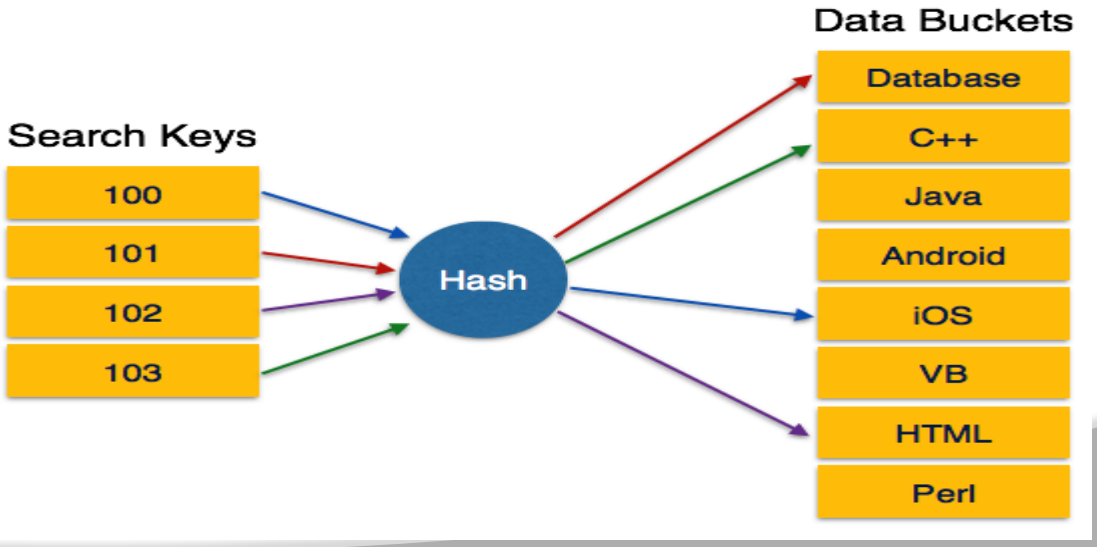
Search − When a record needs to be retrieved, the same hash function can be used to retrieve the address of the bucket where the data is stored.

# STATIC HASHING

- Delete − This is simply a search followed by a deletion operation.

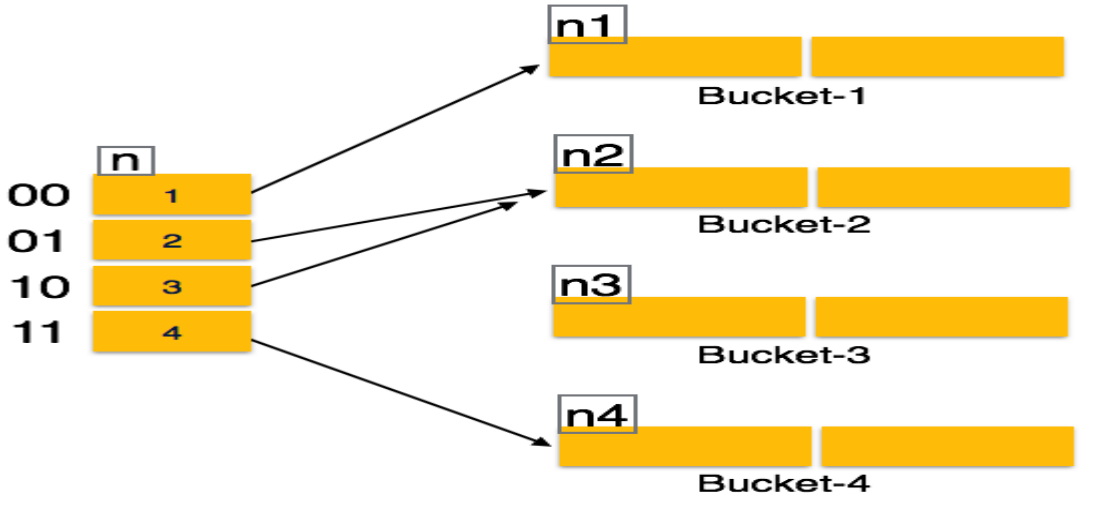
Bucket Overflow

- The condition of bucket-overflow is known as collision. This is a fatal state for any static hash function. In this case, overflow chaining can be used.
- Overflow Chaining − When buckets are full, a new bucket is allocated for the same hash result and is linked after the previous one. This mechanism is called Closed Hashing.



# DYNAMIC HASHING

- The problem with static hashing is that it does not expand or shrink dynamically as the size of the database grows or shrinks.
- Dynamic hashing provides a mechanism in which data buckets are added and removed dynamically and on-demand. Dynamic hashing is also known as extended hashing.
- Hash function, in dynamic hashing, is made to produce a large number of values and only a few are used initially.



# COMPARISON OF ORDERED INDEXING AND HASHING

- Each scheme has advantages in certain situations. And the DBMS implementer could leave the decision to the database designer and provide several methods.
- Normally, the implementer only provides a very limited number of schemes.
- Typically, ordered indexing is used unless it is known in advance that range queries will be infrequent, in which case hashing is used.
- Hash organizations are particularly useful for temporary files created during query processing, if lookups on a key value are required and no ranges queries will be performed.

# COMPARISON OF ORDERED INDEXING AND HASHING

- Index Definition in SQL
- Creating an index:
- CREATE INDEX <index-name> ON <relation-name> (<attribute-list>)

Example:

- CREATE INDEX branch-index ON branch (branch-name)
- Deleting an index:

DROP INDEX <index-name>

# QUERY PROCESSING: OVERVIEW

1. Parsing and translation
2. Optimization
3. Evaluation

- Parsing and translation
  - Translate the query into its internal form. This is then translated into relational algebra.
  - Parser checks syntax, verifies relations.
- Optimization
  - A relational algebra expression may have many equivalent expressions. Each relational algebra operation can be evaluated using one of several different algorithm.
- Evaluation
  - The query-execution engine takes a query-evaluation plan, executes that plan, and returns the answers to the query

# MEASURES OF QUERY COST

- Cost is generally measured as total elapsed time for answering query.
    - Many factors contribute to time cost
    - "disk accesses, CPU, or even network communication.
- Typically disk access is the predominant cost, and is also relatively easy to estimate. Measured by taking into account
    - Number of seeks * average-seek-cost
    - Number of blocks read * average-block-read-cost
    - Number of blocks written * average-block-write-cost
    - "Cost to write a block is greater than cost to read a block – data is read back after being written to ensure that the write was successful

# MEASURES OF QUERY COST

- For simplicity we just use number of block transfers from disk as the cost measure
    - We ignore the difference in cost between sequential and random I/O for simplicity
    - We also ignore CPU costs for simplicity
- Costs depends on the size of the buffer in main memory
    - Having more memory reduces need for disk access
    - Amount of real memory available to buffer depends on other concurrent OS processes, and hard to determine ahead of actual execution
    - We often use worst case estimates, assuming only the minimum amount of memory needed for the operation is available
- Real systems take CPU cost into account, differentiate between sequential and random I/O, and take buffer size into account
- We do not include cost to writing output to disk in our cost formulae