MP:

DBMS:

Database Management System (DBMS)

■ Collection of interrelated data

■ Set of programs to access the data

■ DBMS contains information about a particular enterprise

■ DBMS provides an environment that is both convenient and
efficient to use.

■ Database Applications:

★ Banking: all transactions

★ Airlines: reservations, schedules

★ Universities: registration, grades

★ Sales: customers, products, purchases

★ Manufacturing: production, inventory, orders, supply chain

★ Human resources: employee records, salaries, tax deductions

■ Databases touch all aspects of our lives

Database System Concepts 1.3 ©Silberschatz, Korth and Sudarshan

Purpose of Database System

■ In the early days, database applications were built on top of
file systems

■ Drawbacks of using file systems to store data:

★ Data redundancy and inconsistency

✓ Multiple file formats, duplication of information in different files

★ Difficulty in accessing data

✓ Need to write a new program to carry out each new task

★ Data isolation — multiple files and formats

★ Integrity problems

✓ Integrity constraints (e.g. account balance > 0) become part of program code

✓ Hard to add new constraints or change existing ones

Purpose of Database Systems (Cont.)

■ Drawbacks of using file systems (cont.)

★ Atomicity of updates

✓ Failures may leave database in an inconsistent state with partial updates carried out

✓ E.g. transfer of funds from one account to another should either complete or not happen at all

★ Concurrent access by multiple users

✓ Concurrent accessed needed for performance

✓ Uncontrolled concurrent accesses can lead to inconsistencies

– E.g. two people reading a balance and updating it at the same time

★ Security problems

■ Database systems offer solutions to all the above problems

Levels of Abstraction

■ Physical level describes how a record (e.g., customer) is stored.

■ Logical level: describes data stored in database, and the relationships among the data.

type customer = record

name : string;

street : string;

city : integer;

end;

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

View of Data

An architecture for a database system

Instances and Schemas

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

Data Models

■ A collection of tools for describing

★ data

★ data relationships

★ data semantics

★ data constraints

■ Entity-Relationship model

■ Relational model

■ Other models:

★ object-oriented model

★ semi-structured data models

★ Older models: network model and hierarchical model

Entity-Relationship Model

Example of schema in the entity-relationship model

Entity Relationship Model (Cont.)

■ E-R model of real world

★ Entities (objects)

✓ E.g. customers, accounts, bank branch

★ Relationships between entities

✓ E.g. Account A-101 is held by customer Johnson

✓ Relationship set depositor associates customers with accounts

■ Widely used for database design

★ Database design in E-R model usually converted to design in the relational model (coming up next) which is used for storage and processing

Relational Model

■ Example of tabular data in the relational model

customer-
name

Customer-id

customer-
street

customer-
city

account-
number

Johnson
Smith
Johnson
Jones
Smith

192-83-7465
019-28-3746
192-83-7465
321-12-3123
019-28-3746

Alma

North

Alma

Main

North

Palo Alto

Rye

Palo Alto

Harrison

Rye

A-101

A-215

A-201

A-217

A-201

Attributes

A Sample Relational Database

Data Definition Language (DDL)

■ Specification notation for defining the database schema

★ E.g.

create table account (

account-number char(10),

balance integer)

■ DDL compiler generates a set of tables stored in a data dictionary

■ Data dictionary contains metadata (i.e., data about data)

★ database schema

★ Data storage and definition language

✓ language in which the storage structure and access methods used by the database system are specified

✓ Usually an extension of the data definition language

Data Manipulation Language (DML)

■ Language for accessing and manipulating the data organized by the appropriate data model

★ DML also known as query language

■ Two classes of languages

★ Procedural – user specifies what data is required and how to get those data

★ Nonprocedural – user specifies what data is required without specifying how to get those data

■ SQL is the most widely used query language

SQL

■ SQL: widely used non-procedural language

★ E.g. find the name of the customer with customer-id 192-83-7465

```
select customer.customer-name
from customer
where customer.customer-id = ‘192-83-7465’
```

★ E.g. find the balances of all accounts held by the customer with customer-id 192-83-7465

```
select account.balance
from depositor, account
where depositor.customer-id = ‘192-83-7465’ and

depositor.account-number = account.account-number
```

■ Application programs generally access databases through one of

★ Language extensions to allow embedded SQL

★ Application program interface (e.g. ODBC/JDBC) which allow SQL queries to be sent to a database

Database Users

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

■ Application programmers – interact with system through DML calls

■ Sophisticated users – form requests in a database query language

■ Specialized users – write specialized database applications that do not fit into the traditional data processing framework

■ Naïve users – invoke one of the permanent application programs that have been written previously

★ E.g. people accessing database over the web, bank tellers, clerical staff

Database Administrator

■ Coordinates all the activities of the database system; the
database administrator has a good understanding of the
enterprise's information resources and needs.
■ Database administrator's duties include:

★ Schema definition

★ Storage structure and access method definition

★ Schema and physical organization modification

★ Granting user authority to access the database

★ Specifying integrity constraints

★ Acting as liaison with users

★ Monitoring performance and responding to changes in
requirements

Transaction Management

■ A transaction is a collection of operations that performs a single
logical function in a database application
■ Transaction-management component ensures that the database
remains in a consistent (correct) state despite system failures
(e.g., power failures and operating system crashes) and
transaction failures.
■ Concurrency-control manager controls the interaction among the
concurrent transactions, to ensure the consistency of the
database.

Storage Management

■ 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

Overall System Structure

Application Architectures

Two-tier architecture: E.g. client programs using ODBC/JDBC to communicate with a database

Three-tier architecture: E.g. web-based applications, and applications built using “middleware”

Chapter 2: Entity-Relationship Model

- Entity Sets
- Relationship Sets
- Design Issues
- Mapping Constraints
- Keys
- E-R Diagram
- Extended E-R Features
- Design of an E-R Database Schema
- Reduction of an E-R Schema to Tables

Entity Sets

█ A database can be modeled as:

✓ a collection of entities,

✓ relationship among entities. █ An entity is an object that exists and is distinguishable from other objects. ✓ Example: specific person, company, event, plant

█ Entities have attributes

✓ Example: people have names and addresses

█ An entity set is a set of entities of the same type that share the same properties. ✓ Example: set of all persons, companies, trees, holidays

Entity Sets customer and loan

customer-id customer- customer- customer- loan- amount

name street city number

Attributes

█ An entity is represented by a set of attributes, that is descriptive properties possessed by all members of an entity set.

█ Domain – the set of permitted values for each attribute

█ Attribute types:

✓ Simple and composite attributes. ✓ Single-valued and multi-valued attributes

⚲ E.g. multivalued attribute: phone-numbers

✓ Derived attributes

⚲ Can be computed from other attributes

⚲ E.g. age, given date of birth

Example:

customer = (customer-id, customer-name,

customer-street, customer-city)

loan = (loan-number, amount)

Composite Attributes

Relationship Sets

■ A relationship is an association among several entities

Example:

Hayes depositor A-102

customer entity relationship set account entity

■ A relationship set is a mathematical relation among n ✓ 2 entities,

each taken from entity sets

{(e1

, e2

, ... en

) | e1 ■ E1

, e2 ■ E2

, ..., en ■ En

}

where (e1

, e2

, ..., en

) is a relationship

✓ Example:

(Hayes, A-102) ■ depositor

Relationship Set borrower

Relationship Sets (Cont.)

■ 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). Generally, most relationship sets in a database system are

binary. ■ Relationship sets may involve more than two entity sets.

■ Relationships between more than two entity sets are rare. Most

relationships are binary. (More on this later.)

✓E.g. Suppose employees of a bank may have jobs

(responsibilities) at multiple branches, with different jobs at

different branches. Then there is a ternary relationship set

between entity sets employee, job and branch

Mapping Cardinalities

■ Express the number of entities to which another entity can be associated via a relationship set. ■ Most useful in describing binary relationship sets. ■ For a binary relationship set the mapping cardinality must be

one of the following types:

✓ One to one

✓ One to many

✓ Many to one

✓ Many to many

Mapping Cardinalities

One to one One to many

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

elements in the other set

Mapping Cardinalities

Many to one Many to many

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

elements in the other set

Mapping Cardinalities affect ER Design

■ Can make access-date an attribute of account, instead of a

relationship attribute, if each account can have only one customer

■ I.e., the relationship from account to customer is many to one,

or equivalently, customer to account is one to many

E-R Diagrams

■ Rectangles represent entity sets. ■ Diamonds represent relationship sets. ■ Lines link attributes to entity sets and entity sets to relationship sets. ■ Ellipses represent attributes

■ Double ellipses represent multivalued attributes. ■ Dashed ellipses denote derived attributes. ■ Underline indicates primary key attributes (will study later)

E-R Diagram With Composite, Multivalued, and

Derived Attributes

Relationship Sets with Attributes

Roles

■ Entity sets of a relationship need not be distinct

■ The labels —manager‖ and —worker‖ are called roles; they specify how employee entities interact via the works-for relationship set. ■ Roles are indicated in E-R diagrams by labeling the lines that connect

diamonds to rectangles. ■ Role labels are optional, and are used to clarify semantics of the

relationship

Cardinality Constraints

■ We express cardinality constraints by drawing either a directed

line (⌕), signifying —one,

‖ or an undirected line (—), signifying

—many,

‖ between the relationship set and the entity set. ■ E.g.: One-to-one relationship:

✓ A customer is associated with at most one loan via the relationship

borrower

✓ A loan is associated with at most one customer via borrower

One-To-Many Relationship

■ In the one-to-many relationship a loan is associated with at most

one customer via borrower, a customer is associated with several (including 0) loans via borrower

Many-To-One Relationships

■ In a many-to-one relationship a loan is associated with several

(including 0) customers via borrower, a customer is associated

with at most one loan via borrower

Many-To-Many Relationship

■ A customer is associated with several (possibly 0) loans

via borrower

■ A loan is associated with several (possibly 0) customers

via borrower

Participation of an Entity Set in a

Participation of an Entity Set in a

Relationship Set

■ Total participation (indicated by double line): every entity in the entity set participates in at least one relationship in the relationship set

■ E.g. participation of loan in borrower is total

■ every loan must have a customer associated to it via borrower

■ Partial participation: some entities may not participate in any relationship in the relationship set

■ E.g. participation of customer in borrower is partial

Alternative Notation for Cardinality

Alternative Notation for Cardinality

Limits

■ Cardinality limits can also express participation constraints

Keys

■ A super key of an entity set is a set of one or more attributes whose values uniquely determine each entity. ■ A candidate key of an entity set is a minimal super key

✓ Customer-id is candidate key of customer

✓ account-number is candidate key of account

█ Although several candidate keys may exist, one of the candidate keys is selected to be the primary key.

Keys for Relationship Sets

█ The combination of primary keys of the participating entity sets
forms a super key of a relationship set. ✓ (customer-id, account-number) is the super key of depositor

✓ NOTE: this means a pair of entity sets can have at most one
relationship in a particular relationship set. ⚲ E.g. if we wish to track all access-dates to each account by each customer, we cannot assume a relationship for each access. We can use a multivalued attribute though

█ Must consider the mapping cardinality of the relationship set
when deciding the what are the candidate keys

█ Need to consider semantics of relationship set in selecting the
primary key in case of more than one candidate key

E-R Diagram with a Ternary Relationship

C di lit C t i t T

Cardinality Constraints on Ternary

Relationship

■ We allow at most one arrow out of a ternary (or greater degree)

relationship to indicate a cardinality constraint

■ E.g. an arrow from works-on to job indicates each employee works

on at most one job at any branch. ■ If there is more than one arrow, there are two ways of defining the

meaning. ✓ E.g a ternary relationship R between A, B and C with arrows to B and C

could mean

✓ 1. each A entity is associated with a unique entity from B and C or

✓ 2. each pair of entities from (A, B) is associated with a unique C entity,

and each pair (A, C) is associated with a unique B

✓ Each alternative has been used in different formalisms

✓ To avoid confusion we outlaw more than one arrow

Binary Vs. Non-Binary Relationships

■ Some relationships that appear to be non-binary may be better

represented using binary relationships

✓ E.g. A ternary relationship parents, relating a child to his/her father and

mother, is best replaced by two binary relationships, father and mother

⌕ Using two binary relationships allows partial information (e.g. only

mother being know)

✓ But there are some relationships that are naturally non-binary

⌕ E.g. works-on

Converting Non-Binary Relationships to

Converting Non-Binary Relationships to

Binary Form

■ In general, any non-binary relationship can be represented using binary

relationships by creating an artificial entity set. ✓ Replace R between entity sets A, B and C by an entity set E, and three

relationship sets:

1. RA

, relating E and A 2.RB

, relating E and B

3. RC

, relating E and C

✓ Create a special identifying attribute for E

✓ Add any attributes of R to E

✓ For each relationship (ai , bi , ci) in R, create

1. a new entity ei in the entity set E 2. add (ei , ai) to RA

3. add (ei , bi) to RB 4. add (ei , ci) to RC

Converting Non Binary Relationships

Converting Non-Binary Relationships

(Cont.)

█ Also need to translate constraints

✔ Translating all constraints may not be possible

✔ There may be instances in the translated schema that

cannot correspond to any instance of R

⌕ Exercise: add constraints to the relationships RA

, RB and RC

to

ensure that a newly created entity corresponds to exactly one entity in each of entity sets A, B and C

✔ We can avoid creating an identifying attribute by making E a weak

entity set (described shortly) identified by the three relationship sets

D i l

Design Issues

█ Use of entity sets vs. attributes

Cmhoice mainly depends on the structure of the enterprise being odeled, and on the semantics associated with the attribute in

question. █ Use of entity sets vs. relationship sets

Possible guideline is to designate a relationship set to describe an

action that occurs between entities

█ Binary versus n-ary relationship sets

Although it is possible to replace any nonbinary (n-ary, for n > 2)

relationship set by a number of distinct binary relationship sets, a n- ary relationship set shows more clearly that several entities

participate in a single relationship. ■ Placement of relationship attributes

How about doing an ER design

interactively on the board?

Suggest an application to be modeled.

W k E tit S t

Weak Entity Sets

■ An entity set that does not have a primary key is referred to as a

weak entity set. ■ The existence of a weak entity set depends on the existence of a

identifying entity set

✓ it must relate to the identifying entity set via a total, one-to-many

relationship set from the identifying to the weak entity set

✓ Identifying relationship depicted using a double diamond

■ The discriminator (or partial key) of a weak entity set is the set of

attributes that distinguishes among all the entities of a weak

entity set. ■ The primary key of a weak entity set is formed by the primary key of the strong entity set on which the weak entity set is existence

dependent, plus the weak entity set’s discriminator.

W k E tit S t (C t)

Weak Entity Sets (Cont.)

■ We depict a weak entity set by double rectangles. ■ We underline the discriminator of a weak entity set with a

dashed line. ■ payment-number – discriminator of the payment entity set

■ Primary key for payment – (loan-number, payment-number)

Weak Entity Sets (Cont.)

■ Note: the primary key of the strong entity set is not explicitly

stored with the weak entity set, since it is implicit in the

identifying relationship. ■ If loan-number were explicitly stored, payment could be made a

strong entity, but then the relationship between payment and

loan would be duplicated by an implicit relationship defined by

the attribute loan-number common to payment and loan

M W k E tit S t E l

More Weak Entity Set Examples

■ In a university, a course is a strong entity and a course-offering

can be modeled as a weak entity

■ The discriminator of course-offering would be semester (including

year) and section-number (if there is more than one section)

■ If we model course-offering as a strong entity we would model

course-number as an attribute. Then the relationship with course would be implicit in the course- number attribute

Speciali ation

Specialization

■ Top-down design process; we designate subgroupings within an

entity set that are distinctive from other entities in the set. ■ These subgroupings become lower-level entity sets that have

attributes or participate in relationships that do not apply to the

higher-level entity set. ■ Depicted by a triangle component labeled ISA (E.g. customer —is

a‖ person). ■ Attribute inheritance – a lower-level entity set inherits all the

attributes and relationship participation of the higher-level entity

set to which it is linked.

Specialization Example

Specialization Example

Generalization

Generalization

■ A bottom-up design process – combine a number of entity sets

that share the same features into a higher-level entity set. ■ Specialization and generalization are simple inversions of each

other; they are represented in an E-R diagram in the same way. ■ The terms specialization and generalization are used

interchangeably.

Specialization and Generalization

Specialization and Generalization

(Contd.)

■ Can have multiple specializations of an entity set based on

different features. ■ E.g. permanent-employee vs. temporary-employee, in addition to

officer vs. secretary vs. teller

■ Each particular employee would be

✓ a member of one of permanent-employee or temporary-employee,

✓ and also a member of one of officer, secretary, or teller

■ The ISA relationship also referred to as superclass - subclass

relationship

Design Constraints on a

Design Constraints on a

Specialization/Generalization

■ Constraint on which entities can be members of a given

lower-level entity set. ✓ condition-defined

⌕ E.g. all customers over 65 years are members of senior- citizen entity set; senior-citizen ISA person. ✓ user-defined

■ Constraint on whether or not entities may belong to more than

one lower-level entity set within a single generalization. ✓ Disjoint

⌕ an entity can belong to only one lower-level entity set

⌕ Noted in E-R diagram by writing disjoint next to the ISA

triangle

✓ Overlapping

⌕ an entity can belong to more than one lower-level entity set

Design Constraints on a

Design Constraints on a

Specialization/Generalization (Contd.)

■ Completeness constraint -- specifies whether or not an entity in

the higher-level entity set must belong to at least one of the

lower-level entity sets within a generalization. ✓ total : an entity must belong to one of the lower-level entity sets

✓ partial: an entity need not belong to one of the lower-level entity

sets

Aggregation

Aggregation

■ Consider the ternary relationship works-on, which we saw earlier

■ Suppose we want to record managers for tasks performed by an

employee at a branch

Aggregation (Cont)

Aggregation (Cont.)

■ Relationship sets works-on and manages represent overlapping

information

✓ Every manages relationship corresponds to a works-on relationship

✓ However, some works-on relationships may not correspond to any

manages relationships

⌕ So we can’t discard the works-on relationship

■ Eliminate this redundancy via aggregation

✓ Treat relationship as an abstract entity

✓ Allows relationships between relationships

✓ Abstraction of relationship into new entity

■ Without introducing redundancy, the following diagram represents:

✓ An employee works on a particular job at a particular branch

✓ An employee, branch, job combination may have an associated manager

E R Diagram With Aggregation

E-R Diagram With Aggregation

E R Design Decisions

E-R Design Decisions

■ The use of an attribute or entity set to represent an object. ■ Whether a real-world concept is best expressed by an entity set

or a relationship set. ■ The use of a ternary relationship versus a pair of binary

relationships. ■

The use of a strong or weak entity set. ■ The use of specialization/generalization – contributes to

modularity in the design. ■ The use of aggregation – can treat the aggregate entity set as a single unit without concern for the details of its internal structure.

E-R Diagram for a Banking Enterprise

E-R Diagram for a Banking Enterprise

How about doing another ER design

interactively on the board?

Summary of Symbols Used in E-R

Su a y o Sy bo s Used

Notation

Summary of Symbols (Cont)

Summary of Symbols (Cont.)

Alternative E-R Notations

Alternative E-R Notations

UML

UML

- UML: Unified Modeling Language
- UML has many components to graphically model different
aspects of an entire software system
- UML Class Diagrams correspond to E-R Diagram, but several
differences.

Summary of UML Class Diagram Notation

Summary of UML Class Diagram Notation

UML Class Diagrams (Contd)

UML Class Diagrams (Contd.)

■ Entity sets are shown as boxes, and attributes are shown within the
box, rather than as separate ellipses in E-R diagrams. ■ Binary relationship sets are represented in UML by just drawing a
line connecting the entity sets. The relationship set name is written
adjacent to the line. ■ The role played by an entity set in a relationship set may also be
specified by writing the role name on the line, adjacent to the entity
set. ■ The relationship set name may alternatively be written in a box,
along with attributes of the relationship set, and the box is
connected, using a dotted line, to the line depicting the relationship
set. ■ Non-binary relationships drawn using diamonds, just as in ER
diagrams

UML Class Diagram Notation (Cont)

UML Class Diagram Notation (Cont.)

*Note reversal of position in cardinality constraint depiction

*Generalization can use merged or separate arrows independent

of disjoint/overlapping

overlapping

disjoint

UML Cl Di (C d)

UML Class Diagrams (Contd.)

■ Cardinality constraints are specified in the form l..h, where l denotes

the minimum and h the maximum number of relationships an entity

can participate in. ■ Beware: the positioning of the constraints is exactly the reverse of the

positioning of constraints in E-R diagrams. ■ The constraint 0..* on the E2 side and 0..1 on the E1 side means that

each E2 entity can participate in at most one relationship, whereas

each E1 entity can participate in many relationships; in other words,

the relationship is many to one from E2 to E1. ■ Single values, such as 1 or * may be written on edges; The single

value 1 on an edge is treated as equivalent to 1..1, while * is

equivalent to 0..*.

Reduction of an E-R Schema to Tables

Reduction of an E R Schema to Tables

■ Primary keys allow entity sets and relationship sets to be

expressed uniformly as tables which represent the

contents of the database. ■ A database which conforms to an E-R diagram can be

represented by a collection of tables. ■ For each entity set and relationship set there is a unique

table which is assigned the name of the corresponding

entity set or relationship set. ■ Each table has a number of columns (generally

corresponding to attributes), which have unique names. ■ Converting an E-R diagram to a table format is the basis

for deriving a relational database design from an E-R

diagram.

Representing Entity Sets as Tables

Representing Entity Sets as Tables

■ A strong entity set reduces to a table with the same attributes.

Composite and Multivalued Attributes

Composite and Multivalued Attributes

■ Composite attributes are flattened out by creating a separate attribute

for each component attribute

✓ E.g. given entity set customer with composite attribute name with component attributes first-name and last-name the table corresponding

to the entity set has two attributes

name.first-name and name.last-name

█ A multivalued attribute M of an entity E is represented by a separate

table EM ✓ Table EM has attributes corresponding to the primary key of E and an

attribute corresponding to multivalued attribute M

✓ E.g. Multivalued attribute dependent-names of employee is represented

by a table

employee-dependent-names(employee-id, dname)

✓ Each value of the multivalued attribute maps to a separate row of the

table EM

⌕ E.g., an employee entity with primary key John and

dependents Johnson and Johndotir maps to two rows:

(John, Johnson) and (John, Johndotir)

Representing Weak Entity Sets

Representing Weak Entity Sets

█ A weak entity set becomes a table that includes a column for

the primary key of the identifying strong entity set

Representing Relationship Sets as

p g p

Tables

■ A many-to-many relationship set is represented as a table with
columns for the primary keys of the two participating entity sets,
and any descriptive attributes of the relationship set. ■ E.g.: table for relationship set borrower

Redundancy of Tables

Redundancy of Tables

■ Many-to-one and one-to-many relationship sets that are total
on the many-side can be represented by adding an extra
attribute to the many side, containing the primary key of the
one side
■ E.g.: Instead of creating a table for relationship account- branch, add an attribute branch to the entity set account

Redundancy of Tables (Cont.)

edu da cy o ab es (Co t)

■ For one-to-one relationship sets, either side can be chosen to act

as the —many‖ side

✓ That is, extra attribute can be added to either of the tables

corresponding to the two entity sets

■ If participation is partial on the many side, replacing a table by an

extra attribute in the relation corresponding to the —many‖ side

could result in null values

■ The table corresponding to a relationship set linking a weak

entity set to its identifying strong entity set is redundant. ✓ E.g. The payment table already contains the information that would

appear in the loan-payment table (i.e., the columns loan-number

and payment-number).

Representing Specialization as Tables

Representing Specialization as Tables

■ Method 1:

✓ Form a table for the higher level entity

✓ Form a table for each lower level entity set, include primary key of

higher level entity set and local attributes

table	table attributes
person	name, street, city
customer	name, credit-rating
employee	name, salary

✓ Drawback: getting information about, e.g., employee requires

accessing two tables

Representing Specialization as Tables

p g p
(Cont.)

■ Method 2:

✔ Form a table for each entity set with all local and inherited attributes

table	table attributes
person	name, street, city
customer	name, street, city, credit-rating
employee	name, street, city, salary

✔ If specialization is total, table for generalized entity (person) not required to store information

⚲ Can be defined as a —view‖ relation containing union of specialization tables

⚲ But explicit table may still be needed for foreign key constraints

✔ Drawback: street and city may be stored redundantly for persons who are both customers and employees

Relations Corresponding to

A ti

Aggregation

■ To represent aggregation, create a table containing ■ primary key of the aggregated relationship,

■ the primary key of the associated entity set

■ Any descriptive attributes

Relations Corresponding to

A ti (C t)

Aggregation (Cont.)

■ E.g. to represent aggregation manages between relationship

works-on and entity set manager, create a table

manages(employee-id, branch-name, title, manager-name)

■ Table works-on is redundant provided we are willing to store

null values for attribute manager-name in table manages

End of Chapter 2

E-R Diagram for Exercise 2.10

g

E-R Diagram for Exercise 2.15

g

E-R Diagram for Exercise 2.22

g

E-R Diagram for Exercise 2.15

g

Existence Dependencies

http://slidepdf.com/reader/full/02-entity-relationship-model 73/73

■ If the existence of entity x depends on the existence of

entity y, then x is said to be existence dependent on y. ✓ y is a dominant entity (in example below, loan)

✓ x is a subordinate entity (in example below, payment)

loan loan-payment payment

If a loan entity is deleted, then all its associated payment entities

must be deleted also.

SELECT DISTINCT JOB FROM EMP;

2. List employees whose salary is more than 3000 after giving 20% increment.

SELECT ENAME,SAL,SAL+(SAL*0.2) FROM SCOTT.EMP WHERE (SAL+(SAL*0.2))>3000

3. List employees details who does not belong to dept 20 or 30.

SELECT ENAME,SAL,DEPTNO FROM SCOTT.EMP WHERE DEPTNO NOT IN (20,30) ;

4. List employees name,salary whose annual salary does not fall from 20000 to 40000.

SELECT ENAME,SAL,SAL*12 FROM SCOTT.EMP WHERE (SAL*12) NOT BETWEEN

20000 AND 40000

5. List all employees who join before 1981.

SELECT ENAME,SAL,HIREDATE FROM SCOTT.EMP WHERE

TO_CHAR(HIREDATE,'YYYY')<1981

6. List all emp nos,emp names,job except 'president and 'Manager'.

SELECT ENAME,SAL,HIREDATE,JOB FROM SCOTT.EMP WHERE JOB NOT IN

('PRESIDENT','MANAGER');

7. Display the name of employee who does not work under any manager.

SELECT ENAME,SAL,HIREDATE,JOB FROM SCOTT.EMP WHERE MGR IS NULL;

8. Display all emp names whose name is 5 characters long.

SELECT ENAME FROM SCOTT.EMP WHERE ENAME LIKE '_____';

9. Change the name of emp SCOTT to E.F.CODD.

UPDATE EMP SET ENAME='E.F.CODD' WHERE ENAME ='SCOTT';

10. Display the name of employees in dept 30 who can earn a commision.

SELECT ENAME FROM SCOTT.EMP WHERE deptno=30 and comm is not null;

11. Find the total salaries given to Managers.

Select sum(sal),job from scott.emp where job='MANAGER' GROUP BY JOB;

12. Create a view which hides sal n comission details.

CREATE VIEW EMP_VIEW AS SELECT ENAME,EMPNO,JOB,DEPTNO,HIREDATE,MGR

Introduction to SQL

Exercises

3.1 Write the following queries in SQL, using the university schema. (We sug-

gest you actually run these queries on a database, using the sample data

that we provide on the Web site of the book, db-book.com. Instructions for

setting up a database, and loading sample data, are provided on the above

Web site.)

a. Find the titles of courses in the Comp. Sci. department that have 3

credits.

b. Find the IDs of all students who were taught by an instructor named

Einstein; make sure there are no duplicates in the result.

c. Find the highest salary of any instructor.

d. Find all instructors earning the highest salary (there may be more

than one with the same salary).

e. Find the enrollment of each section that was offered in Autumn 2009.

f. Find the maximum enrollment, across all sections, in Autumn 2009.

g. Find the sections that had the maximum enrollment in Autumn 2009.

Answer:

a. Find the titles of courses in the Comp. Sci. department that have 3

credits.

select title

from course

where dept name = ’Comp. Sci.’

and credits = 3

b. Find the IDs of all students who were taught by an instructor named

Einstein; make sure there are no duplicates in the result.

This query can be answered in several different ways. One way is as

follows.

select distinct student.ID

from (student join takes using(ID))

join (instructor join teaches using(ID))

using(course id, sec id, semester, year)

where instructor.name = ’Einstein’

As an alternative to th join .. using syntax above the query can be

written by enumerating relations in the from clause, and adding the

corresponding join predicates on ID, course id,section id,semester, and

year to the where clause.

Note that using natural join in place of join .. using would result in

equating student ID with instructor ID, which is incorrect.

c. Find the highest salary of any instructor.

select max(salary)

from instructor

d. Find all instructors earning the highest salary (there may be more

than one with the same salary).

select ID, name

from instructor

where salary = (select max(salary) from instructor)

e. Find the enrollment of each section that was offered in Autumn 2009.

One way of writing the query is as follows.

select course id, sec id, count(ID)

```
from section natural join takes
where semester = ’Autumn’
and year = 2009
group by course id, sec id
```

Note that if a section does not have any students taking it, it would not appear in the result. One way of ensuring such a section appears with a count of 0 is to replace natural join by the natural left outer join operation, covered later in Chapter 4. Another way is to use a subquery in the select clause, as follows.

```
select course id, sec id,
(select count(ID)
from takes
where takes.year = section.year
and takes.semester = section.semester
and takes.course id = section.course id
and takes.section id = section.section id)
from section
where semester = ’Autumn’
and year = 2009
```

Note that if the result of the subquery is empty, the aggregate function count returns a value of 0.

f. Find the maximum enrollment, across all sections, in Autumn 2009.

One way of writing this query is as follows:

```
select max(enrollment)
from (select count(ID) as enrollment
from section natural join takes
where semester = ’Autumn’
and year = 2009
```

group by course id, sec id)

As an alternative to using a nested subquery in the from clause, it is

possible to use a with clause, as illustrated in the answer to the next

part of this question.

A subtle issue in the above query is that if no section had any enroll-

ment, the answer would be empty, not 0. We can use the alternative

using a subquery, from the previous part of this question, to ensure

the count is 0 in this case.

g. Find the sections that had the maximum enrollment in Autumn 2009.

The following answer uses a with clause to create a temporary view,

simplifying the query.

```
with sec enrollment as (
select course id, sec id, count(ID) as enrollment
from section natural join takes
where semester = ’Autumn’
and year = 2009
group by course id, sec id)
select course id, sec id
from sec enrollment
where enrollment = (select max(enrollment) from sec enrollment)
```

It is also possible to write the query without the with clause, but the

subquery to find enrollment would get repeated twice in the query.

3.2 Suppose you are given a relation grade points(grade, points), which provides

a conversion from letter grades in the takes relation to numeric scores; for

example an “A” grade could be specified to correspond to 4 points, an “A−”

to 3.7 points, a “B+” to 3.3 points, a “B” to 3 points, and so on. The grade

points earned by a student for a course offering (section) is defined as the

number of credits for the course multiplied by the numeric points for the grade that the student received.

Given the above relation, and our university schema, write each of the following queries in SQL. You can assume for simplicity that no takes tuple has the null value for grade.

a. Find the total grade-points earned by the student with ID 12345, across all courses taken by the student.

b. Find the grade-point average (GPA) for the above student, that is, the total grade-points divided by the total credits for the associated courses.

c. Find the ID and the grade-point average of every student.

Answer:

a. Find the total grade-points earned by the student with ID 12345, across all courses taken by the student.

```
select sum(credits * points)
from (takes natural join course) natural join grade points
whereID = ’12345’
```

One problem with the above query is that if the student has not taken any course, the result would not have any tuples, whereas we would expect to get 0 as the answer. One way of fixing this problem is to use the natural left outer join operation, which we study later in Chapter 4. Another way to ensure that we get 0 as the answer, is to the following query:

```
(select sum(credits * points)
from (takes natural join course) natural join grade points
where ID = ’12345’)
union
(select 0
from student
where takes.ID = ’12345’ and
not exists ( select * from takes where takes.ID = ’12345’))
```

As usual, specifying join conditions can be specified in the where clause instead of using the natural join operation or the join .. using operation.

b. Find the grade-point average (GPA) for the above student, that is, the total grade-points divided by the total credits for the associated courses.

```
select sum(credits * points)/sum(credits) as GPA
from (takes natural join course) natural join grade points
where ID = ’12345’
```

As before, a student who has not taken any course would not appear in the above result; we can ensure that such a student appears in the result by using the modified query from the previous part of this question. However, an additional issue in this case is that the sum of credits would also be 0, resulting in a divide by zero condition. In fact, the only meaningful way of defining the GPA in this case is to define it as null. We can ensure that such a student appears in the result with a null GPA by adding the following union clause to the above query.

```
union
(select null as GPA
from student
where takes.ID = ’12345’ and
not exists ( select * from takes where takes.ID = ’12345’))
```

Other ways of ensuring the above are discussed later in the solution to Exercise 4.5.

c. Find the ID and the grade-point average of every student.

```
select ID, sum(credits * points)/sum(credits) as GPA
from (takes natural join course) natural join grade points
group by ID
```

Again, to handle students who have not taken any course, we would have to add the following union clause:

```
union
(select ID, null as GPA
from student
where not exists ( select * from takes where takes.ID = student.ID))
```

3.3

3.4 Write the following inserts, deletes or updates in SQL, using the university schema.

a. Increase the salary of each instructor in the Comp. Sci. department by 10%.

b. Delete all courses that have never been offered (that is, do not occur in the section relation).

c. Insert every student whose tot cred attribute is greater than 100 as an instructor in the same department, with a salary of $10,000.

Answer:

a. Increase the salary of each instructor in the Comp. Sci. department by 10%.

```
update instructor
set salary = salary * 1.10
where dept name = ’Comp. Sci.’
```

b. Delete all courses that have never been offered (that is, do not occur in the section relation).

```
delete from course
where course id not in
(select course id from section)
```

c. Insert every student whose tot cred attribute is greater than 100 as an instructor in the same department, with a salary of $10,000.

```
insert into instructor
select ID, name, dept name, 10000
from student
where tot cred > 100
```

3.5 Consider the insurance database of Figure ??, where the primary keys

are underlined. Construct the following SQL queries for this relational

database.

a. Find the total number of people who owned cars that were involved

in accidents in 1989.

b. Add a new accident to the database; assume any values for required

attributes.

c. Delete the Mazda belonging to “John Smith”.

Answer: Note: The participated relation relates drivers, cars, and accidents.

a. Find the total number of people who owned cars that were involved

in accidents in 1989.

Note: this is not the same as the total number of accidents in 1989.

We must count people with several accidents only once.

```
select count (distinct name)
from accident, participated, person
where accident.report number = participated.report number
and participated.driver id = person.driver id
and date between date ’1989-00-00’ and date ’1989-12-31’
```

person (driver id, name, address)

car (license, model, year)

accident (report number, date, location)

owns (driver id, license)

participated (driver id, car, report number, damage amount)

Figure ??. Insurance database.

b. Add a new accident to the database; assume any values for required

attributes.

We assume the driver was “Jones,” although it could be someone

else. Also, we assume “Jones” owns one Toyota. First we must find the license of the given car. Then the participated and accidentrelations

must be updated in order to both record the accident and tie it to the

given car. We assume values “Berkeley” for location, ’2001-09-01’ for

date and date, 4007 for report number and 3000 for damage amount.

insert into accident

values (4007, ’2001-09-01’, ’Berkeley’)

insert into participated

select o.driver id, c.license, 4007, 3000

from person p, owns o, car c

where p.name = ’Jones’ and p.driver id = o.driver id and

o.license = c.license and c.model = ’Toyota’

c. Delete the Mazda belonging to “John Smith”.

Since model is not a key of the car relation, we can either assume

that only one of John Smith’s cars is a Mazda, or delete all of John

Smith’s Mazdas (the query is the same). Again assume name is a key

for person.

delete car

where model = ’Mazda’ and license in

(select license

from person p, owns o

where p.name = ’John Smith’ and p.driver id = o.driver id)

Note: The owns, accident and participated records associated with the

Mazda still exist.

3.6 Suppose that we have a relation marks(ID, score) and we wish to assign

grades to students based on the score as follows: grade F if score < 40,

grade C if 40 ≤ score < 60, grade B if 60 ≤ score < 80, and grade A if 80 ≤

score. Write SQL queries to do the following:

a. Display the grade for each student, based on the marks relation.

b. Find the number of students with each grade.

Answer:

a. Display the grade for each student, based on the marks relation.

```
select ID,
case
when score < 40 then ’F’
when score < 60 then ’C’
when score < 80 then ’B’
else ’A’
end
from marks
```

b. Find the number of students with each grade.

```
with grades as
(
select ID,
case
when score < 40 then ’F’
when score < 60 then ’C’
when score < 80 then ’B’
else ’A’
end as grade
from marks
)
select grade, count(ID)
from grades
group by grade
```

As an alternative, the with clause can be removed, and instead the definition of grades can be made a subquery of the main query.

3.7 The SQL like operator is case sensitive, but the lower() function on strings can be used to perform case insensitive matching. To show how, write a query that finds departments whose names contain the string “sci” as a substring, regardless of the case.

Answer:

```
select dept name
from department
where lower(dept name) like ’%sci%’
```

3.8 Consider the SQL query

branch(branch name, branch city, assets)

customer (customer name, customer street, customer city)

loan (loan number, branch name, amount)

borrower (customer name, loan number)

account (account number, branch name, balance)

depositor (customer name, account number)

Figure 3.1 Banking database for Exercises 3.8 and 3.15.

```
select p.a1
from p, r1, r2
where p.a1 = r1.a1 or p.a1 = r2.a1
```

Under what conditions does the preceding query select values of p.a1 that are either in r1 or in r2? Examine carefully the cases where one of r1 or r2 may be empty.

Answer: The query selects those values of p.a1 that are equal to some value of r1.a1 or r2.a1 if and only if both r1 and r2 are non-empty. If one or both of r1 and r2 are empty, the cartesian product of p, r1 and r2 is empty, hence the result of the query is empty. Of course if p itself is empty, the result is as expected, i.e. empty.

3.9 Consider the bank database of Figure 3.19, where the primary keys are un-derlined. Construct the following SQL queries for this relational database.

a. Find all customers of the bank who have an account but not a loan.

b. Find the names of all customers who live on the same street and in the same city as “Smith”.

c. Find the names of all branches with customers who have an account in the bank and who live in “Harrison”.

Answer:

a. Find all customers of the bank who have an account but not a loan.

```
(select customer name
from depositor)
except
(select customer name
from borrower)
```

The above selects could optionally have distinct specified, without changing the result of the query.

b. Find the names of all customers who live on the same street and in the same city as “Smith”.

One way of writing the query is as follows.

```
select F.customer name
from customer F join customer S using(customer street, customer city)
where S.customer name = ’Smith’
```

The join condition could alternatively be specified in the where clause, instead of using bf join .. using.

c. Find the names of all branches with customers who have an account in the bank and who live in “Harrison”.

```
select distinct branch name
from account natural join depositor natural join customer
where customer city = ’Harrison’
```

As usual, the natural join operation could be replaced by specifying join conditions in the where clause.

3.10 Consider the employee database of Figure ??, where the primary keys are

underlined. Give an expression in SQL for each of the following queries.

a. Find the names and cities of residence of all employees who work

for First Bank Corporation.

b. Find the names, street addresses, and cities of residence of all em-

ployees who work for First Bank Corporation and earn more than

$10,000.

c. Find all employees in the database who do not work for First Bank

Corporation.

d. Find all employees in the database who earn more than each em-

ployee of Small Bank Corporation.

e. Assume that the companies may be located in several cities. Find all

companies located in every city in which Small Bank Corporation is

located.

f. Find the company that has the most employees.

g. Find those companies whose employees earn a higher salary, on

average, than the average salary at First Bank Corporation.

Answer:

employee (employee name, street, city)

works (employee name, company name, salary)

company (company name, city)

manages (employee name, manager name)

Figure 3.20. Employee database.

Exercises 15

a. Find the names and cities of residence of all employees who work

for First Bank Corporation.

select e.employee name, city

from employee e, works w

where w.company name = ’First Bank Corporation’ and

w.employee name = e.employee name

b. Find the names, street address, and cities of residence of all em-

ployees who work for First Bank Corporation and earn more than

$10,000.

If people may work for several companies, the following solution

will only list those who earn more than $10,000 per annum from

“First Bank Corporation” alone.

select *

from employee

where employee name in

(select employee name

from works

where company name = ’First Bank Corporation’ and salary > 10000)

As in the solution to the previous query, we can use a join to solve

this one also.

c. Find all employees in the database who do not work for First Bank

Corporation.

The following solution assumes that all people work for exactly one

company.

select employee name

from works

where company name 6= ’First Bank Corporation’

If one allows people to appear in the database (e.g. in employee) but

not appear in works, or if people may have jobs with more than one

company, the solution is slightly more complicated.

select employee name

from employee

where employee name not in

(select employee name

```
from works
where company name = ’First Bank Corporation’)
```

d. Find all employees in the database who earn more than each em-
ployee of Small Bank Corporation.

The following solution assumes that all people work for at most one
company.

```
select employee name
from works
where salary > all
(select salary
from works
where company name = ’Small Bank Corporation’)
```

If people may work for several companies and we wish to consider
the total earnings of each person, the problem is more complex. It
can be solved by using a nested subquery, but we illustrate below
how to solve it using the with clause.

```
with emp total salary as
(select employee name, sum(salary) as total salary
from works
group by employee name
)
select employee name
from emp total salary
where total salary > all
(select total salary
from emp total salary, works
where works.company name = ’Small Bank Corporation’ and
emp total salary.employee name = works.employee name
)
```

e. Assume that the companies may be located in several cities. Find all companies located in every city in which Small Bank Corporation is

located.

The simplest solution uses the contains comparison which was in-

cluded in the original System R Sequel language but is not present

in the subsequent SQL versions.

```
select T.company name
from company T
where (select R.city
from company R
where R.company name = T.company name)
contains
(select S.city
from company S
where S.company name = ’Small Bank Corporation’)
```

Below is a solution using standard SQL.

```
select S.company name
from company S
where not exists ((select city
from company
where company name = ’Small Bank Corporation’)
except
(select city
from company T
where S.company name = T.company name))
```

f. Find the company that has the most employees.

```
select company name
```

```
from works
group by company name
having count (distinct employee name) >= all
(select count (distinct employee name)
from works
group by company name)
```

g. Find those companies whose employees earn a higher salary, on

average, than the average salary at First Bank Corporation.

```
select company name
from works
group by company name
having avg (salary) > (select avg (salary)
from works
where company name = ’First Bank Corporation’)
```

3.11 Consider the relational database of Figure ??. Give an expression in SQL for

each of the following queries.

a. Modify the database so that Jones now lives in Newtown.

b. Give all managers of First Bank Corporation a 10 percent raise unless

the salary becomes greater than $100,000; in such cases, give only a

3 percent raise.

Answer:

a. Modify the database so that Jones now lives in Newtown.

The solution assumes that each person has only one tuple in the

employee relation.

```
update employee
set city = ’Newton’
where person name = ’Jones’
```

b. Give all managers of First Bank Corporation a 10-percent raise unless

the salary becomes greater than $100,000; in such cases, give only a

3-percent raise.

```
update works T
set T.salary = T.salary * 1.03
where T.employee name in (select manager name
from manages)
and T.salary * 1.1 > 100000
and T.company name = ’First Bank Corporation’
update works T
set T.salary = T.salary * 1.1
where T.employee name in (select manager name
from manages)
and T.salary * 1.1 <= 100000
and T.company name = ’First Bank Corporation’
```

The above updates would give different results if executed in the

opposite order. We give below a safer solution using the case state-

ment.

```
update works T
set T.salary = T.salary ∗
(case
when (T.salary ∗ 1.1 > 100000) then 1.03
else 1.1
)
where T.employee name in (select manager name
from manages) and
T.company name = ’First Bank Corporation’
```

OS:

Module 1: Operating system Overview

• 1.1 Introduction, Objectives, Functions and Evolution of Operating System

• 1.2 Operating system structures: Layered, Monolithic and Microkernel • 1.3 Linux Kernel, Shell and System Calls

● What is an operating System?

In the Computer System (comprises of Hardware and software), Hardware can only understand machine code (in the form of 0 and 1) which doesn't make any sense to a naive user.
We need a system which can act as an intermediary and manage all the processes and resources present in the system.

An Operating System can be defined as an interface between user and hardware. It is responsible for the execution of all the processes, Resource Allocation, CPU management, File Management and many other tasks.
The purpose of an operating system is to provide an environment in which a user can execute programs in convenient and efficient manner.
The operating system is a system program that serves as an interface between the computing system and the end-user. Operating systems create an environment where the user can run any programs or communicate with software or applications in a comfortable and well-organized way.

Furthermore, an operating is a software program that manages and controls the execution of application programs, software resources and computer hardware. It also helps manage the software/hardware resource, such as file management, memory management, input/ output and many peripheral devices like a disk drive, printers, etc.

● Explain different objectives of operating System?

There are three objectives of operating system:

A) Convenience: An OS makes a computer more convenient to use.

B) Efficiency: An OS allows the computer system resources to be used in an
efficient manner.

C) Ability to evolve: An OS should be constructed in such a way as to permit the
effective development, testing and introduction of new system functions at
the same time without interfering with service.

● Explain different Functions of operating System?

The most important functions of operating system are:

A) Process Management: The operating system is responsible for the following activities in connection with the process management

● Scheduling processes and threads on the CPU

● Creating and deleting both user and system processes

● Suspending and resuming processes

● Providing mechanism for process synchronization

● Providing mechanisms for process communication

B) Memory Management: The operating system is responsible for the following activities in connection with the memory management

● Keeping track of which parts of memory is currently being used and by whom

● Deciding which processes and data to move into and out of memory

● Allocating and de allocating memory space as needed.

C) File Management: The operating system is responsible for the following activities in connection with the file management

● Creating and deleting files

● Creating and deleting directories to organize files

● Supporting primitives for manipulating files and directories

● Mapping files onto secondary storage

● Backing up the files on non-volatile storage media

D) Device Management: The operating system is responsible for the following activities in connection with the device management

● Keeps tracks of all devices connected to system.

● designates a program responsible for every device known as the Input/Output controller

● Decides which process gets access to a certain device and for how long.

● Allocates devices in an effective and efficient way.

● De allocates devices when they are no longer required.

E) Security: The operating system uses password protection to protect user data and similar other techniques. it also prevents unauthorized access to programs and user data.

F) Booting of computer: The operating system boots the computer.

G) Control over system performance –

Monitors overall system health to help improve performance.

● Records the response time between service requests and system response to

have a complete view of the system health.

● Explain different types of operating System?

A) Simple Batch Systems ● In this type of system, there is no direct interaction between user and the computer. ● The user has to submit a job (written on cards or tape) to a computer operator. ● Then computer operator places a batch of several jobs on an input device. ● Jobs are batched together by type of languages and requirement. ● Then a special program, the monitor, manages the execution of each program in the

batch. ● The monitor is always in the main memory and available for execution.

B) Multiprogramming Operating System

● Multiprogramming is an extension to the batch processing where the CPU is kept always

busy.

● Each process needs two types of system time: CPU time and IO time.

● In multiprogramming environment, for the time a process does its I/O, The CPU can start the

execution of other processes. Therefore, multiprogramming improves the efficiency of the

system. ● In Non-multiprogrammed system, there are moments when CPU sits idle and

does not do any work. ● In Multiprogramming system, CPU will never be idle and keeps on processing.

Time Sharing Systems are very similar to Multiprogramming batch systems. In fact

time sharing systems are an extension of multiprogramming systems.

In Time sharing systems the prime focus is on minimizing the response time, while in

multiprogramming the prime focus is to maximize the CPU usage.

C) Multiprocessing Operating System

● In Multiprocessing, Parallel computing is achieved. There are more than one processors

present in the system which can execute more than one process at the same time. This will

increase the throughput of the system.

D) Real Time Operating System

● In Real Time systems, each job carries a certain deadline within which the Job is supposed to

be completed, otherwise the huge loss will be there or even if the result is produced then it

will be completely useless.

● The Application of a Real Time system exists in the case of military applications, if you want

to drop a missile then the missile is supposed to be dropped with certain precision.

E) Distributed Operating System

● The motivation behind developing distributed operating systems is the availability of powerful and inexpensive microprocessors and advances in communication technology.

● These advancements in technology have made it possible to design and develop distributed systems comprising of many computers that are inter connected by communication networks. The main benefit of distributed systems is its low price/performance ratio.

F) Clustered Systems ● Like parallel systems, clustered systems gather together multiple CPUs to accomplish computational work. ● Clustered systems differ from parallel systems, however, in that they are composed of two or more individual systems coupled together. ● The definition of the term clustered is not concrete; the general accepted definition is

that clustered computers share storage and are closely linked via LAN networking. ● Clustering is usually performed to provide high availability. G) Handheld Systems

● Handheld systems include Personal Digital Assistants(PDAs), such as Cellular Telephones with connectivity to a network such as the Internet. They are usually of limited size due to which most handheld devices have a small amount of memory, include slow processors, and feature small display screens.

● Explain services provided by operating System?

An Operating System supplies different kinds of services to both the users and to the programs as well. It also provides application programs (that run within an Operating system) an environment to execute it freely.

Here is a list of common services offered by an almost all operating systems:

A) User InterfaceUsually Operating system comes in three forms or types.

Depending on the interface their types have been further subdivided. These are: ● Command line interface ● Batch based interface ● Graphical User Interface

The command line interface (CLI) usually deals with using text commands and a technique for entering those commands. The batch interface (BI): commands and directives are used to manage those commands that are entered into files and those files get executed. Another type is the graphical user interface (GUI): which is a window system with a pointing device (like mouse or trackball) to point to the I/O, choose from menus driven interface and to make choices viewing from a number of lists and a keyboard to entry the texts.

B) Program Execution

The operating system must have the capability to load a program into memory and execute that program. Furthermore, the program must be able to end its execution, either normally or abnormally / forcefully.

C) File system manipulation

Programs need has to be read and then write them as files and directories. File handling portion of operating system also allows users to create and delete files by specific name along with extension, search for a given file and / or list file information. Some programs comprise of permissions management for allowing or denying access to files or directories based on file ownership.

D) Input / Output Operations

A program which is currently executing may require I/O, which may involve file or other I/O device. For efficiency and protection, users cannot directly govern the I/O devices. So, the OS provide a means to do I/O Input / Output operation which means read or write operation with any file.

E) Communication

Process needs to swap over information with other process. Processes executing on same computer system or on different computer systems can communicate using operating system support. Communication between two processes can be done using shared memory or via message passing.

F) Resource Allocation

When multiple jobs running concurrently, resources must need to be allocated to each of them. Resources can be CPU cycles, main memory storage, file storage and I/O devices. CPU scheduling routines are used here to establish how best the CPU can be used.

G) Error Detection

Errors may occur within CPU, memory hardware, I/O devices and in the user program. For each type of error, the OS takes adequate action for ensuring correct and consistent computing.

H) Accounting

This service of the operating system keeps track of which users are using how much and what kinds of computer resources have been used for accounting or simply to accumulate usage statistics.

I) Security and protection

Protection includes in ensuring all access to system resources in a controlled manner. For making a system secure, the user needs to authenticate him or her to the system before using

(usually via login ID and password).

Explain the operating System Structure

The operating system structure are categorized as-

A)Monolithic Approach

The monolithic operating system is also known as the monolithic kernel. This is an old

type of operating system. They were used to perform small tasks like batch processing,

time sharing tasks in banks. Monolithic kernel acts as a virtual machine which controls

all hardware parts. It is different than microkernel which has limited tasks.

A microkernel is divided into two parts i.e. kernel space and user space. Both these

parts communicate with each other through IPC (Inter-process communication).

Microkernel advantage is that if one server fails then other server takes control of it.

Operating systems which use monolithic architecture were first time used in the

1970's.

The monolithic operating system is a very basic operating system in which file

management, memory management, device management, and process management is

directly controlled within the kernel. All these components like file management,

memory management etc. are located within the kernel.

Advantages of Monolithic Kernel – ● One of the major advantage of having monolithic kernel is that it provides CPU

scheduling, memory management, file management and other operating system

functions through system calls. ● The other one is that it is a single large process running entirely in a single address

space. ● It is a single static binary file. Example of some Monolithic Kernel based OSs are:

Unix, Linux, Open VMS, XTS-400, z/TPF.

Disadvantages of Monolithic Kernel – ● One of the major disadvantage of monolithic kernel is that, if anyone service fails it

leads to entire system failure. ● If user has to add any new service. User needs to modify entire operating system.

B) Layered Approach

An OS can be broken into pieces and retain much more control on system. In this structure the OS is broken into number of layers (levels). The bottom layer (layer 0) is the hardware and the topmost layer (layer N) is the user interface. These layers are so designed that each layer uses the functions of the lower level layers only. This simplifies

the debugging process as if lower level layers are debugged and an error occurs during debugging then the error must be on that layer only as the lower level layers have already been debugged.
The main disadvantage of this structure is that at each layer, the data needs to be modified and passed on which adds overhead to the system. Moreover careful planning of the layers is necessary as a layer can use only lower level layers. UNIX is an example of this structure.

C) Microkernel approach
This structure designs the operating system by removing all non-essential components from the kernel and implementing them as system and user programs. This result in a smaller kernel called the micro-kernel.
Advantages of this structure are that all new services need to be added to user space and does not require the kernel to be modified. Thus it is more secure and reliable as if a service fails then rest of the operating system remains untouched. Mac OS is an example of this type of OS.
Modular structure or approach:
It is considered as the best approach for an OS. It involves designing of a modular kernel. The kernel has only set of core components and other services are added as dynamically loadable modules to the kernel either during run time or boot time. It resembles layered structure due to the fact that each kernel has defined and protected

interfaces but it is more flexible than the layered structure as a module can call any other module.
For example Solaris OS is organized as shown in the figure.

Explain the differences between monolithic and microkernel System
Structure

Basis of

Comparison

Monolithic Kernel MicroKernel

Execution Style All processes are executed under
the kernel space in privileged
mode.

Only the most important processes
take place in the Kernel space. All
other processes are executed in the
user space.

Size Kernel size is bigger when
compared to Microkernel.

Kernel size is smaller with respect
to the monolithic kernel.

Speed It provides faster execution of

processes.

Process execution is slower.

Stability A single process crash will cause
the entire system to crash.

A single process crash will have no
impact on other processes.

Inter-Process
Communication

Use signals and sockets to achieve

interprocess communication.

Use messaging queues to achieve

inter process communication.

Extensibility Difficult to extend. Easily extensible.

Maintainability Maintenance is more time and

resource consuming.

Easily maintainable

Debug Harder to debug Easier to debug

Security Less Secure. More Secure

Example Linux Mac OS

System Calls

To understand system calls, first one needs to understand the difference

between kernel mode and user mode of a CPU. Every modern operating system

supports these two modes.

Modes supported by the operating system

Kernel Mode

● When CPU is in kernel mode, the code being executed can access any memory

address and any hardware resource. ● Hence kernel mode is a very privileged and powerful mode.

● If a program crashes in kernel mode, the entire system will be halted.

User Mode

● When CPU is in user mode, the programs don't have direct access to memory

and hardware resources. ● In user mode, if any program crashes, only that particular program is halted. ● That means the system will be in a safe state even if a program in user mode

crashes. ● Hence, most programs in an OS run in user mode.

When a program in user mode requires access to RAM or a hardware resource, it must ask the kernel to provide access to that resource. This is done via something called

a system call. When a program makes a system call, the mode is switched from user mode to kernel

mode. This is called a context switch. Then the kernel provides the resource which the program requested. After that, another

context switch happens which results in change of mode from kernel mode back to user

mode.

A system call is a mechanism that provides the interface between a process and the

operating system. It is a programmatic method in which a computer program requests a

service from the kernel of the OS.

System call offers the services of the operating system to the user programs via API

(Application Programming Interface). System calls are the only entry points for the

kernel system.

Why do you need System Calls in OS?

Following are situations which need system calls in OS: ● Reading and writing from files demand system calls. ● If a file system wants to create or delete files, system calls are required. ● System calls are used for the creation and management of new processes. ● Network connections need system calls for sending and receiving packets. ● Access to hardware devices like scanner, printer, need a system call.

Types of System calls

Here are the five types of system calls used in OS: ● Process Control ● File Management ● Device Management ● Information Maintenance ● Communications

Process Control

This system calls perform the task of process creation, process termination, etc.

Functions: ● End and Abort ● Load and Execute ● Create Process and Terminate Process ● Wait and Signed Event ● Allocate and free memory

File Management

File management system calls handle file manipulation jobs like creating a file, reading,

and writing, etc.

Functions: ● Create a file ● Delete file ● Open and close file ● Read, write, and reposition

● Get and set file attributes

Device Management

Device management does the job of device manipulation like reading from device

buffers, writing into device buffers, etc.

Functions ● Request and release device ● Logically attach/ detach devices ● Get and Set device attributes

Information Maintenance

It handles information and its transfer between the OS and the user program.

Functions: ● Get or set time and date ● Get process and device attributes

Communication:

These types of system calls are specially used for interprocess communications.

Functions: ● Create, delete communications connections ● Send, receive message ● Help OS to transfer status information

● Attach or detach remote devices

Linux Operating system

LINUX is an operating system or a kernel distributed under an open-source

license. Its functionality list is quite like UNIX. The kernel is a program at

the heart of the Linux operating system that takes care of fundamental

stuff, like letting hardware communicate with software.

Linux architecture

Linux Kernel

The main purpose of a computer is to run a predefined sequence of instructions, known

as a program. A program under execution is often referred to as a process. Now, most

special purpose computers are meant to run a single process, but in a sophisticated

system such a general purpose computer, are intended to run many processes

simultaneously. Any kind of process requires hardware resources such are Memory,

Processor time, Storage space, etc.

In a General Purpose Computer running many processes simultaneously, we need a

middle layer to manage the distribution of the hardware resources of the computer

efficiently and fairly among all the various processes running on the computer. This

middle layer is referred to as the kernel. Basically the kernel virtualizes the common

hardware resources of the computer to provide each process with its own virtual

resources. This makes the process seem as it is the sole process running on the machine.

The kernel is also responsible for preventing and mitigating conflicts between different

processes.

This schematically represented below:

Figure: Virtual Resources for each Process

The Core Subsystems of the Linux Kernel are as follows:

1. The Process Scheduler
2. The Memory Management Unit (MMU)
3. The Virtual File System (VFS)
4. The Networking Unit
5. Inter-Process Communication Unit

Figure: The Linux Kernel

For the purpose of this article we will only be focussing on the 1st three important subsystems of the Linux Kernel.

The basic functioning of each of the 1st three subsystems is elaborated below: ● The Process Scheduler: This kernel subsystem is responsible for fairly distributing the CPU time among all the processes running on the system simulteneously. ● The Memory Management Unit: This kernel sub-unit is responsible for proper distribution of the memory resources among the various processes running on the system. The MMU does more than just simply provide separate virtual address spaces for each of the processes. ● The Virtual File System: This subsystem is responsible for providing a unified interface to access stored data across different filesystems and physical storage media.

Write a short note on shell

SHELL is a program which provides the interface between the user and an operating system. When the user logs in OS starts a shell for user. It gathers input from you and executes programs based on that input. When a program finishes executing, it displays that program's output.

Shell is an environment in which we can run our commands, programs, and shell scripts. There are different flavors of a shell, just as there are different flavors of operating systems. Each flavor of shell has its own set of recognized commands and functions.

Shell Prompt

The prompt, $, which is called the command prompt, is issued by the shell. While the prompt is displayed, you can type a command.

Shell reads your input after you press Enter. It determines the command you want executed by looking at the first word of your input. A word is an unbroken set of characters. Spaces and tabs separate words.

Types of Shell: ● The C Shell –

Denoted as csh

Bill Joy created it at the University of California at Berkeley. It incorporated features such as aliases and command history. It includes helpful programming features like built-in arithmetic and C-like expression syntax.

In C shell:

Command full-path name is /bin/csh,

Non-root user default prompt is hostname %,

Root user default prompt is hostname #. ● The Bourne Shell –

Denoted as sh

It was written by Steve Bourne at AT&T Bell Labs. It is the original UNIX shell. It is faster and more preferred. It lacks features for interactive use like the ability to recall previous commands. It also lacks built-in arithmetic and logical expression handling. It is default shell for Solaris OS.

For the Bourne shell the:

Command full-path name is /bin/sh and /sbin/sh,

Non-root user default prompt is $,

Root user default prompt is #. ● The Korn Shell

It is denoted as ksh

It Was written by David Korn at AT&T Bell LabsIt is a superset of the Bourne shell.So it supports everything in the Bourne shell.It has interactive features. It includes features like built-in arithmetic and C-like arrays, functions, and string-manipulation facilities.It is faster than C shell. It is compatible with script written for C shell.

For the Korn shell the:

Command full-path name is /bin/ksh,

Non-root user default prompt is $,

Root user default prompt is #.

Shell Scripts

The basic concept of a shell script is a list of commands, which are listed in the order of execution. A good shell script will have comments, preceded by # sign, describing the steps.

There are conditional tests, such as value A is greater than value B, loops allowing us to go through massive amounts of data, files to read and store data, and variables to read and store data, and the script may include functions.

Example Script

Assume we create a test.sh script. Note all the scripts would have the .sh extension.

Before you add anything else to your script, you need to alert the system that a shell script is being started. This is done using the shebang construct. For example −

```
#!/bin/sh
```

This tells the system that the commands that follow are to be executed by the Bourne shell. It's called a shebang because the # symbol is called a hash, and the ! symbol is called a bang. To create a script containing these commands, you put the shebang line first and then add the commands −

```
#!/bin/bash

pwd
ls
```

The shell script is now ready to be executed −

```
$./test.sh
```

Upon execution, you will receive the following result −

```
/home/amrood
index.htm unix-basic_utilities.htm unix-directories.htm
test.sh unix-communication.htm unix-environment.htm
```

Note − To execute a program available in the current directory,

Module 2: Process and Process management

A process is a program in execution which then forms the basis of all computation. The process is not as same as program code but a lot more than it. A process is an 'active' entity as opposed to the program which is considered to be a 'passive' entity.

Process Architecture

Here, is an Architecture diagram of the Process

 Stack: The Stack stores temporary data like function parameters, returns addresses, and local variables. Heap Allocates memory, which may be processed during its run time. Data: It contains the variable. Text: Text Section includes the current activity, which is represented by the value of the Program Counter.

Process States:

A process state is a condition of the process at a specific instant of time. It also defines

the current position of the process.

There are mainly seven stages of a process which are:

 New: The new process is created when a specific program calls from secondary

memory/ hard disk to primary memory/ RAM a

 Ready: In a ready state, the process should be loaded into the primary memory,

which is ready for execution. Waiting: The process is waiting for the allocation of CPU time and other

resources for execution. Executing: The process is an execution state. Blocked: It is a time interval when a process is waiting for an event like I/O

operations to complete. Suspended: Suspended state defines the time when a process is ready for

execution but has not been placed in the ready queue by OS. Terminated: Terminated state specifies the time when a process is terminated

After completing every step, all the resources are used by a process, and memory

becomes free.

Process Control Block (PCB)

Every process is represented in the operating system by a process control block, which

is also called a task control block.

Here, are important components of PCB

 Process state: A process can be new, ready, running, waiting, etc.

 Program counter: The program counter lets you know the address of the next

instruction, which should be executed for that process. CPU registers: This component includes accumulators, index and general-

purpose registers, and information of condition code. CPU scheduling information: This component includes a process priority,

pointers for scheduling queues, and various other scheduling parameters. Accounting and business information: It includes the amount of CPU and time

utilities like real time used, job or process numbers, etc. Memory-management information: This information includes the value of the

base and limit registers, the page, or segment tables. This depends on the memory system, which is used by the operating system. I/O status information: This block includes a list of open files, the list of I/O

devices that are allocated to the process.

What is Process Scheduling?

The act of determining which process is in the ready state, and should be moved to

the running state is known as Process Scheduling.

The prime aim of the process scheduling system is to keep the CPU busy all the time and

to deliver minimum response time for all programs. For achieving this, the scheduler

must apply appropriate rules for swapping processes IN and OUT of CPU.

Scheduling fell into one of the two general categories: Non Pre-emptive Scheduling: When the currently executing process gives up

the CPU voluntarily. Pre-emptive Scheduling: When the operating system decides to favour another

process, pre-empting the currently executing process.

What are Scheduling Queues?

 All processes, upon entering into the system, are stored in the Job Queue. Processes in the Ready state are placed in the Ready Queue. Processes waiting for a device to become available are placed in Device Queues. There are unique device queues available for each I/O device.

A new process is initially put in the Ready queue. It waits in the ready queue until it is

selected for execution (or dispatched). Once the process is assigned to the CPU and is

executing, one of the following several events can occur: The process could issue an I/O request, and then be placed in the I/O queue. The process could create a new subprocess and wait for its termination. The process could be removed forcibly from the CPU, as a result of an interrupt,

and be put back in the ready queue.

Types of Schedulers

There are three types of schedulers available:

1. Long Term Scheduler

2. Short Term Scheduler

3. Medium Term Scheduler

Long Term Scheduler

Long term scheduler runs less frequently. Long Term Schedulers decide which program

must get into the job queue. From the job queue, the Job Processor, selects processes

and loads them into the memory for execution. Primary aim of the Job Scheduler is to

maintain a good degree of Multiprogramming. An optimal degree of Multiprogramming

means the average rate of process creation is equal to the average departure rate of processes from the execution memory.

Short Term Scheduler

This is also known as CPU Scheduler and runs very frequently. The primary aim of this scheduler is to enhance CPU performance and increase process execution rate.

Medium Term Scheduler

This scheduler removes the processes from memory (and from active contention for the CPU), and thus reduces the degree of multiprogramming. At some later time, the process can be reintroduced into memory and its execution van be continued where it left off. This scheme is called swapping. The process is swapped out, and is later swapped in, by the medium term scheduler.

What is Context Switch?

1. Switching the CPU to another process requires saving the state of the old process and loading the saved state for the new process. This task is known as a Context Switch. 2. The context of a process is represented in the Process Control Block(PCB) of a

process; it includes the value of the CPU registers, the process state and memory-management information. When a context switch occurs, the Kernel saves the

context of the old process in its PCB and loads the saved context of the new process scheduled to run.

3. Context switch time is pure overhead, because the system does no useful work while switching. Its speed varies from machine to machine, depending on the memory speed, the number of registers that must be copied, and the existence of special instructions(such as a single instruction to load or store all registers). Typical speeds range from 1 to 1000 microseconds.

4. Context Switching has become such a performance bottleneck that programmers are using new structures(threads) to avoid it whenever and wherever possible.

CPU Scheduling: Scheduling Criteria

There are many different criteria to check when considering the "best" scheduling algorithm, they are:

CPU Utilization

To make out the best use of the CPU and not to waste any CPU cycle, the CPU would be working most of the time(Ideally 100% of the time). Considering a real system, CPU usage should range from 40% (lightly loaded) to 90% (heavily loaded.)

Throughput

It is the total number of processes completed per unit of time or rather says the total amount of work done in a unit of time. This may range from 10/second to 1/hour depending on the specific processes.

Turnaround Time

It is the amount of time taken to execute a particular process, i.e. The interval from the time of submission of the process to the time of completion of the process(Wall clock time).

Waiting Time

The sum of the periods spent waiting in the ready queue amount of time a process has been waiting in the ready queue to acquire get control on the CPU.

Load Average

It is the average number of processes residing in the ready queue waiting for their turn to get into the CPU.

Response Time

Amount of time it takes from when a request was submitted until the first response is produced. Remember, it is the time till the first response and not the completion of process execution (final response).

In general CPU utilization and Throughput are maximized and other factors are reduced for proper optimization.

Scheduling Algorithms

To decide which process to execute first and which process to execute last to achieve maximum CPU utilization, computer scientists have defined some algorithms, they are:

First Come First Serve Scheduling

In the "First come first serve" scheduling algorithm, as the name suggests,

the process which arrives first, gets executed first, or we can say that the process which

requests the CPU first, gets the CPU allocated first. First Come First Serve, is just like FIFO(First in First out) Queue data structure,

where the data element which is added to the queue first, is the one who leaves

the queue first. This is used in Batch Systems. It's easy to understand and implement programmatically, using a Queue data structure, where a new process enters through the tail of the queue, and the

scheduler selects process from the head of the queue. A perfect real life example of FCFS scheduling is buying tickets at ticket

counter. Calculating Average Waiting Time

For every scheduling algorithm, Average waiting time is a crucial parameter to judge

it's performance.

AWT or Average waiting time is the average of the waiting times of the processes in the

queue, waiting for the scheduler to pick them for execution.

Lower the Average Waiting Time, better the scheduling algorithm.

Consider the processes P1, P2, P3, P4 given in the below table, arrives for execution in

the same order, with Arrival Time 0, and given Burst Time, let's find the average

waiting time using the FCFS scheduling algorithm.

The average waiting time will be 18.75 ms

For the above given proccesses, first P1 will be provided with the CPU resources, Hence, waiting time for P1 will be 0

 P1 requires 21 ms for completion, hence waiting time for P2 will be 21 ms

 Similarly, waiting time for process P3 will be execution time of P1 + execution

time for P2, which will be (21 + 3) ms = 24 ms. For process P4 it will be the sum of execution times of P1, P2 and P3. The GANTT chart above perfectly represents the waiting time for each process

Problems with FCFS Scheduling

Below we have a few shortcomings or problems with the FCFS scheduling algorithm:

1. It is Non Pre-emptive algorithm, which means the process priority doesn't

matter.

If a process with very least priority is being executed, more like daily routine

backup process, which takes more time, and all of a sudden some other high

priority process arrives, like interrupt to avoid system crash, the high priority

process will have to wait, and hence in this case, the system will crash, just

because of improper process scheduling.

2. Not optimal Average Waiting Time.

3. Resources utilization in parallel is not possible, which leads to Convoy Effect, and hence poor resource(CPU, I/O etc) utilization

Shortest Job First(SJF) Scheduling

Shortest Job First scheduling works on the process with the shortest burst time or duration first. This is the best approach to minimize waiting time. This is used in Batch Systems. It is of two types:

1. Non Pre-emptive

2. Pre-emptive

 To successfully implement it, the burst time/duration time of the processes

should be known to the processor in advance, which is practically not feasible all

the time. This scheduling algorithm is optimal if all the jobs/processes are available at the

same time. (either Arrival time is 0 for all, or Arrival time is same for all)

Non Pre-emptive Shortest Job First

Consider the below processes available in the ready queue for execution, with arrival

time as 0 for all and given burst times.

As you can see in the GANTT chart above, the process P4 will be picked up first as it has

the shortest burst time, then P2, followed by P3 and at last P1. We scheduled the same set of processes using the First come first serve algorithm in the

previous tutorial, and got average waiting time to be 18.75 ms, whereas with SJF, the

average waiting time comes out 4.5 ms. Problem with Non Pre-emptive SJF

If the arrival time for processes are different, which means all the processes are not

available in the ready queue at time 0, and some jobs arrive after some time, in such

situation, sometimes process with short burst time have to wait for the current

process's execution to finish, because in Non Pre-emptive SJF, on arrival of a process

with short duration, the existing job/process's execution is not halted/stopped to

execute the short job first.

This leads to the problem of Starvation, where a shorter process has to wait for a long

time until the current longer process gets executed. This happens if shorter jobs keep

coming, but this can be solved using the concept of aging.

Pre-emptive Shortest Job First

In Preemptive Shortest Job First Scheduling, jobs are put into ready queue as they

arrive, but as a process with short burst time arrives, the existing process is

preempted or removed from execution, and the shorter job is executed first.

The average waiting time will be,((5-3)+(6-2)+(12-1))/4=8.75

The average waiting time for preemptive shortest job first scheduling is less than both,non preemptive SJF scheduling and FCFS scheduling

As you can see in the GANTT chart above, as P1 arrives first, hence it's execution starts immediately, but just after 1 ms, process P2 arrives with a burst time of 3 ms which is less than the burst time of P1, hence the process P1(1 ms done, 20 ms left) is preemptied and process P2 is executed.

As P2 is getting executed, after 1 ms, P3 arrives, but it has a burst time greater than that of P2, hence execution of P2 continues. But after another millisecond, P4 arrives with a burst time of 2 ms, as a result P2(2 ms done, 1 ms left) is preemptied and P4 is executed.

After the completion of P4, process P2 is picked up and finishes, then P2 will get executed and at last P1. The Pre-emptive SJF is also known as Shortest Remaining Time First, because at any given point of time, the job with the shortest remaining time is executed first.

Priority CPU Scheduling

In the Shortest Job First scheduling algorithm, the priority of a process is generally the inverse of the CPU burst time, i.e. the larger the burst time the lower is the priority of that process.

In case of priority scheduling the priority is not always set as the inverse of the CPU burst time, rather it can be internally or externally set, but yes the scheduling is done on the basis of priority of the process where the process which is most urgent is processed first, followed by the ones with lesser priority in order.

Processes with same priority are executed in FCFS manner.

The priority of process, when internally defined, can be decided based on memory requirements, time limits ,number of open files, ratio of I/O burst to CPU burst etc.

Whereas, external priorities are set based on criteria outside the operating system, like the importance of the process, funds paid for the computer resource use, makrte factor etc.

Types of Priority Scheduling Algorithm

Priority scheduling can be of two types:

1. Preemptive Priority Scheduling: If the new process arrived at the ready queue has a higher priority than the currently running process, the CPU is preempted, which means the processing of the current process is stoped and the incoming new process with higher priority gets the CPU for its execution.

2. Non-Preemptive Priority Scheduling: In case of non-preemptive priority

scheduling algorithm if a new process arrives with a higher priority than the

current running process, the incoming process is put at the head of the ready

queue, which means after the execution of the current process it will be

processed.

Example of Priority Scheduling Algorithm

Consider the below table fo processes with their respective CPU burst times and the

priorities.

As you can see in the GANTT chart that the processes are given CPU time just on the

basis of the priorities.

Problem with Priority Scheduling Algorithm

In priority scheduling algorithm, the chances of indefinite blocking or starvation. A process is considered blocked when it is ready to run but has to wait for the CPU as

some other process is running currently.

But in case of priority scheduling if new higher priority processes keeps coming in the

ready queue then the processes waiting in the ready queue with lower priority may

have to wait for long durations before getting the CPU for execution.

Using Aging Technique with Priority Scheduling

To prevent starvation of any process, we can use the concept of aging where we keep

on increasing the priority of low-priority process based on the its waiting time.

For example, if we decide the aging factor to be 0.5 for each day of waiting, then if a

process with priority 20(which is comparitively low priority) comes in the ready queue.

After one day of waiting, its priority is increased to 19.5 and so on.

Doing so, we can ensure that no process will have to wait for indefinite time for getting

CPU time for processing.

Round Robin Scheduling

Round Robin(RR) scheduling algorithm is mainly designed for time-sharing systems.

This algorithm is similar to FCFS scheduling, but in Round Robin(RR) scheduling,

preemption is added which enables the system to switch between processes. A fixed time is allotted to each process, called a quantum, for execution. Once a process is executed for the given time period that process is preempted

and another process executes for the given time period. Context switching is used to save states of preempted processes. This algorithm is simple and easy to implement and the most important is thing is this algorithm is starvation-free as all processes get a fair share of CPU. It is important to note here that the length of time quantum is generally from 10

to 100 milliseconds in length.

Some important characteristics of the Round Robin(RR) Algorithm are as follows:

1. Round Robin Scheduling algorithm resides under the category of Preemptive

Algorithms.

2. This algorithm is one of the oldest, easiest, and fairest algorithm.

3. This Algorithm is a real-time algorithm because it responds to the event within a

specific time limit.

4. In this algorithm, the time slice should be the minimum that is assigned to a

specific task that needs to be processed. Though it may vary for different

operating systems.

5. This is a hybrid model and is clock-driven in nature.

6. This is a widely used scheduling method in the traditional operating system.

Important terms

1. Completion Time

It is the time at which any process completes its execution.

2. Turn Around Time

This mainly indicates the time Difference between completion time and arrival

time. The Formula to calculate the same is: Turn Around Time = Completion

Time – Arrival Time

3. Waiting Time(W.T):

It Indicates the time Difference between turn around time and burst time.

And is calculated as Waiting Time = Turn Around Time – Burst Time

Let us now cover an example for the same:

In the above diagram, arrival time is not mentioned so it is taken as 0 for all

processes.

Note: If arrival time is not given for any problem statement then it is taken as 0 for all

processes; if it is given then the problem can be solved accordingly.

Explanation

The value of time quantum in the above example is 5.Let us now calculate the Turn

around time and waiting time for the above example :

Processes

Burst

Time

Turn Around Time

Turn Around Time =

Completion Time – Arrival

Time

Waiting Time

Waiting Time = Turn

Around Time – Burst

Time

P1 21 32-0=32 32-21=11

P2 3 8-0=8 8-3=5

P3 6 21-0=21 21-6=15

P4 2 15-0=15 15-2=13

Average waiting time is calculated by adding the waiting time of all processes and then dividing them by no.of processes.

average waiting time = waiting time of all processes/ no.of processes

average waiting time=11+5+15+13/4 = 44/4= 11ms

What are Threads?

Thread is an execution unit that consists of its own program counter, a stack, and a set of registers where the program counter mainly keeps track of which instruction to execute next, a set of registers mainly hold its current working variables, and a stack mainly contains the history of execution

Threads are also known as Lightweight processes. Threads are a popular way to improve the performance of an application through parallelism. Threads are mainly

used to represent a software approach in order to improve the performance of an
operating system just by reducing the overhead thread that is mainly equivalent to a
classical process.
The CPU switches rapidly back and forth among the threads giving the illusion that the
threads are running in parallel.
As each thread has its own independent resource for process execution; thus Multiple
processes can be executed parallelly by increasing the number of threads.
It is important to note here that each thread belongs to exactly one process and outside
a process no threads exist. Each thread basically represents the flow of control
separately. In the implementation of network servers and web servers threads have
been successfully used. Threads provide a suitable foundation for the parallel execution
of applications on shared-memory multiprocessors.
The given below figure shows the working of a single-threaded and a multithreaded
process:

Before moving on further let us first understand the difference between a process and a
thread.

Process Thread

A Process simply means any program in execution. Thread simply means a segment of a process.

The process consumes more resources Thread consumes fewer resources.

The process requires more time for creation. Thread requires comparatively less time for

creation than process.

The process is a heavyweight process Thread is known as a lightweight process

The process takes more time to terminate The thread takes less time to terminate.

Processes have independent data and code segments A thread mainly shares the data segment, code segment, files, etc. with its peer threads.

The process takes more time for context switching. The thread takes less time for context switching.

Communication between processes needs more time
as compared to thread.

Communication between threads needs less time
as compared to processes.

For some reason, if a process gets blocked then the
remaining processes can continue their execution

In case if a user-level thread gets blocked, all of its
peer threads also get blocked.

Advantages of Thread
Some advantages of thread are given below:

1. Responsiveness
2. Resource sharing, hence allowing better utilization of resources.
3. Economy. Creating and managing threads becomes easier.
4. Scalability. One thread runs on one CPU. In Multithreaded processes, threads can be distributed over a series of processors to scale.
5. Context Switching is smooth. Context switching refers to the procedure followed by the CPU to change from one task to another.
6. Enhanced Throughput of the system. Let us take an example for this: suppose a process is divided into multiple threads, and the function of each thread is considered as one job, then the number of jobs completed per unit of time increases which then leads to an increase in the throughput of the system.

Types of Thread
There are two types of threads:
1. User Threads
2. Kernel Threads

User threads are above the kernel and without kernel support. These are the threads that application programmers use in their programs.
Kernel threads are supported within the kernel of the OS itself. All modern OSs support kernel-level threads, allowing the kernel to perform multiple simultaneous tasks and/or to service multiple kernel system calls simultaneously.
Let us now understand the basic difference between User level Threads and Kernel level threads:

User Level threads Kernel Level Threads

These threads are implemented by users. These threads are implemented by Operating

systems

These threads are not recognized by operating
systems,

These threads are recognized by operating systems,

In User Level threads, the Context switch
requires no hardware support.

In Kernel Level threads, hardware support is
needed.

These threads are mainly designed as
dependent threads.

These threads are mainly designed as independent
threads.

In User Level threads, if one user-level thread
performs a blocking operation then the entire
process will be blocked.

On the other hand, if one kernel thread performs a

blocking operation then another thread can

continue the execution.

Example of User Level threads: Java thread,

POSIX threads. Example of Kernel level threads: Window Solaris.

Implementation of User Level thread is done by

a thread library and is easy.

While the Implementation of the kernel-level thread

is done by the operating system and is complex.

Multithreading Models

The user threads must be mapped to kernel threads, by one of the following strategies: Many to One Model One to One Model

 Many to Many Model

Many to One Model In the many to one model, many user-level threads are all mapped onto a single

kernel thread. Thread management is handled by the thread library in user space, which is

efficient in nature. In this case, if user-level thread libraries are implemented in the operating

system in some way that the system does not support them, then the Kernel

threads use this many-to-one relationship model.

One to One Model The one to one model creates a separate kernel thread to handle each and every

user thread. Most implementations of this model place a limit on how many threads can be

created. Linux and Windows from 95 to XP implement the one-to-one model for threads. This model provides more concurrency than that of many to one Model.

Many to Many Model The many to many model multiplexes any number of user threads onto an equal

or smaller number of kernel threads, combining the best features of the one-to-

one and many-to-one models. Users can create any number of threads. Blocking the kernel system calls does not block the entire process. Processes can be split across multiple processors.

Module 3: Process Synchronization and

Deadlocks

Inter Process Communication in Operating System (OS)

There are 2 types of process –

- o Independent Processes – Processes which do not share data with other processes .
- o Cooperating Processes – Processes that shares data with other processes.

Cooperating process require Interprocess communication (IPC) mechanism.

Inter Process Communication is the mechanism by which cooperating process share data and information.

There are 2 ways by which Interprocess communication is achieved

- o Shared memory
- o Message Parsing

Shared Memory

1. A particular region of memory is shared between cooperating process.
2. Cooperating process can exchange information by reading and writing data to this shared region.
3. It's faster than Memory Parsing, as Kernel is required only once, that is, setting up a shared memory . After That, kernel assistance is not required.

Message Parsing

1. Communication takes place by exchanging messages directly between cooperating process.
2. Easy to implement
3. Useful for small amount of data.
4. Implemented using System Calls, so takes more time than Shared Memory.

Process Synchronization in Operating System

There are two ways any process can execute –

 In Concurrent Execution – the CPU scheduler switches rapidly between

processes. A process is stopped at any points and the processor is assigned to

another instruction execution. Here, only one instruction is executed at a time. Parallel execution – 2 or more instructions of different process execute

simultaneously on different processing cores.

Approaches for Inter-Process Communication

Here, are few important methods for interprocess communication:

Pipes

Pipe is widely used for communication between two related processes. This is a half-

duplex method, so the first process communicates with the second process. However, in

order to achieve a full-duplex, another pipe is needed.

Message Passing:

It is a mechanism for a process to communicate and synchronize. Using message

passing, the process communicates with each other without resorting to shared

variables.

IPC mechanism provides two operations: Send (message)- message size fixed or variable

 Received (message)

Message Queues:

A message queue is a linked list of messages stored within the kernel. It is identified by

a message queue identifier. This method offers communication between single or

multiple processes with full-duplex capacity.

Direct Communication:

In this type of inter-process communication process, should name each other explicitly.

In this method, a link is established between one pair of communicating processes, and

between each pair, only one link exists.

Indirect Communication:

Indirect communication establishes like only when processes share a common mailbox

each pair of processes sharing several communication links. A link can communicate

with many processes. The link may be bi-directional or unidirectional.

Shared Memory:

Shared memory is a memory shared between two or more processes that are

established using shared memory between all the processes. This type of memory

requires to protected from each other by synchronizing access across all the processes.

FIFO:

Communication between two unrelated processes. It is a full-duplex method, which

means that the first process can communicate with the second process, and the opposite

can also happen.

Why is Process Synchronisation Important

When several threads (or processes) share data, running in parallel on different cores ,

then changes made by one process may override changes made by another process

running parallel. Resulting in inconsistent data. So, this requires processes to be

synchronized, handling system resources and processes to avoid such situation is

known as Process Synchronization.

Classic Banking Example –

 Consider your bank account has 5000$. You try to withdraw 4000$ using net banking and simultaneously try to

withdraw via ATM too. For Net Banking at time t = 0ms bank checks you have 5000$ as balance and

you’re trying to withdraw 4000$ which is lesser than your available balance. So,

it lets you proceed further and at time t = 1ms it connects you to server to

transfer the amount Imagine, for ATM at time t = 0.5ms bank checks your available balance which

currently is 5000$ and thus let’s you enter ATM password and withdraw

amount.

 At time t = 1.5 ms ATM dispenses the cash of 4000$ and at time t = 2 net banking

transfer is complete of 4000$.

Effect on the system

Now, due to concurrent access and processing time that computer takes in both ways

you were able to withdraw 3000$ more than your balance. In total 8000$ were taken

out and balance was just 5000$.

How to solve this Situation

To avoid such situations process synchronisation is used, so another concurrent

process P2 is notified of existing concurrent process P1 and not allowed to go through

as there is P1 process which is running and P2 execution is only allowed once P1

completes.

Race Condition

In an Operating System, we have a number of processes and these

processes require a number of resources. Now, think of a situation

where we have two processes and these processes are using the

same variable "a". They are reading the variable and then updating

the value of the variable and finally writing the data in the

memory. SomeProcess(){

...

read(a) //instruction 1

a = a + 5 //instruction 2

write(a) //instruction 3

...

}

In the above, you can see that a process after doing some

operations will have to read the value of "a", then increment the

value of "a" by 5 and at last write the value of "a" in the memory. Now, we have two processes P1 and P2 that needs to be executed. Let's take the following two cases and also assume that the value of "a" is 10 initially.

1. In this case, process P1 will be executed fully (i.e. all the three

instructions) and after that, the process P2 will be executed. So, the process P1 will first read the value of "a" to be 10 and

then increment the value by 5 and make it to 15. Lastly, this

value will be updated in the memory. So, the current value of "a" is 15. Now, the process P2 will read the value i.e. 15, increment with 5(15+5 = 20) and finally write it to the

memory i.e. the new value of "a" is 20. Here, in this case, the

final value of "a" is 20. 2. In this case, let's assume that the process P1 starts executing. So, it reads the value of "a" from the memory and that value is

10(initial value of "a" is taken to be 10). Now, at this time, context switching happens between process P1 and P2(learn

more about context switching from here). Now, P2 will be in the running state and P1 will be in the waiting state and the

context of the P1 process will be saved. As the process P1

didn't change the value of "a", so, P2 will also read the value

of "a" to be 10. It will then increment the value of "a" by 5 and

make it to 15 and then save it to the memory. After the

execution of the process P2, the process P1 will be resumed

and the context of the P1 will be read. So, the process P1 is

having the value of "a" as 10(because P1 has already executed

the instruction 1). It will then increment the value of "a" by 5

and write the final value of "a" in the memory i.e. a = 15. Here, the final value of "a" is 15. In the above two cases, after the execution of the two processes P1

and P2, the final value of "a" is different i.e. in 1st case it is 20 and

in 2nd case, it is 15. What's the reason behind this?

The processes are using the same resource here i.e. the variable

"a". In the first approach, the process P1 executes first and then the

process P2 starts executing. But in the second case, the process P1

was stopped after executing one instruction and after that the

process P2 starts executing. And here both the processes are

dealing on the same resource i.e. variable "a" at the same time. Here, the order of execution of processes changes the output. All

these processes are in a race to say that their output is correct. This

is called a race condition.

Critical Section

The code in the above part is accessed by all the process and this

can lead to data inconsistency. So, this code should be placed in

the critical section. The critical section code can be accessed by

only one process at a time and no other process can access that

critical section code. All the shared variables or resources are

placed in the critical section that can lead to data inconsistency.

All the Critical Section problems need to satisfy the following three

conditions:

 Mutual Exclusion: If a process is in the critical section, then other processes shouldn't be allowed to enter into the critical section at that time i.e. there must be some mutual

exclusion between processes. Progress: If in the critical section, there is no process that is

being executed, then other processes that need to go in the

critical section and have a finite time can enter into the

critical section.

 Bounded Waiting: There must be some limit to the

number of times a process can go into the critical section i.e. there must be some upper bound. If no upper bound is there

then the same process will be allowed to go into the critical

section again and again and other processes will never get a

chance to get into the critical section. So, in order to remove the problem of race condition, there must

be synchronization between various processes present in the

system for its execution otherwise, it may lead to data

inconsistency. Mutex in Operating system (OS)

Mutex lock is essentially a variable that is binary in nature that provides code wise

functionality for mutual exclusion. At times, there may be multiple threads that may be

trying to access same resource like memory or I/O etc. To make sure that there is no

overriding. Mutex provides a locking mechanism. Only one thread at a time can take the

ownership of a mutex and apply the lock. Once it done utilising the resource and it may

release the mutex lock.

Mutex Highlights

1. Mutex is Binary in nature
2. Operations like Lock and Release are possible
3. Mutex is for Threads, while Semaphores are for processes.

4. Mutex works in user-space and Semaphore for kernel
5. Mutex provides locking mechanism
6. A thread may acquire more than one mutex
7. Binary Semaphore and mutex are different

Semaphore

A semaphore is a variable that indicates the number of resources

that are available in a system at a particular time and this semaphore variable is generally used to achieve the process

synchronization. It is generally denoted by "S". You can use any

other variable name of your choice. A semaphore uses two functions i.e. wait() and signal(). Both

these functions are used to change the value of the semaphore but

the value can be changed by only one process at a particular time

and no other process should change the value simultaneously. The wait() function is used to decrement the value of the

semaphore variable "S" by one if the value of the semaphore

variable is positive. If the value of the semaphore variable is 0, then no operation will be performed. wait(S) {

while (S == 0); //there is ";" sign here

S--;

}

The signal() function is used to increment the value of the

semaphore variable by one. signal(S) {

S++;

}

Types of Semaphore

There are two types of semaphores:

 Binary Semaphores: In Binary semaphores, the value of

the semaphore variable will be 0 or 1. Initially, the value of

semaphore variable is set to 1 and if some process wants to

use some resource then the wait() function is called and the

value of the semaphore is changed to 0 from 1. The process

then uses the resource and when it releases the resource then

the signal() function is called and the value of the semaphore

variable is increased to 1. If at a particular instant of time, the

value of the semaphore variable is 0 and some other process

wants to use the same resource then it has to wait for the

release of the resource by the previous process. In this way, process synchronization can be achieved. Counting Semaphores: In Counting semaphores, firstly, the semaphore variable is initialized with the number of

resources available. After that, whenever a process needs

some resource, then the wait() function is called and the

value of the semaphore variable is decreased by one. The process then uses the resource and after using the resource, the signal() function is called and the value of the semaphore

variable is increased by one. So, when the value of the

semaphore variable goes to 0 i.e all the resources are taken by

the process and there is no resource left to be used, then if

some other process wants to use resources then that process

has to wait for its turn. In this way, we achieve the process

synchronization.

What is the difference between Mutex and Semaphore in Operating System

Mutex

Example – Imagine mutex as a key to a single toilet. Only one person can be inside

the toilet at a time and he would want to lock the toilet and others waiting for toilet

access must wait for person already inside to release the toilet. When finished the

person gives the key to next person in the queue. Mutex is for thread. Mutex is essentially atomic and singular in nature. Mutex is binary in nature. Operations like Lock and Release are possible

 Mutex works in userspace

 Only one thread can acquire a mutex at a time

 Mutex is an object

Semaphore

Example – Imagine a bathroom with 4 identical toilets here, identical keys can open

the bathrooms. Here as many as 4 people can use the bathroom simultaneously and

they wait and signal one another about the occupancy of the bathrooms

 Semaphore is for processes

 Semaphores are also atomic but not singular in nature

 Semaphores are of two types that are binary and counting

 Semaphores work in kernel space

 Only one process can acquire binary semaphore at a time but multiple processes can

simultaneously acquire semaphore in case of counting semaphore

 Semaphore is an integer variable

Producer-Consumer problem

The Producer-Consumer problem is a classic problem this is used for multi-process synchronization i.e. synchronization between

more than one processes. In the producer-consumer problem, there is one Producer that is

producing something and there is one Consumer that is consuming

the products produced by the Producer. The producers and

consumers share the same memory buffer that is of fixed-size. The job of the Producer is to generate the data, put it into the

buffer, and again start generating data. While the job of the

Consumer is to consume the data from the buffer. What's the problem here?

The following are the problems that might occur in the Producer- Consumer:

 The producer should produce data only when the buffer is not

full. If the buffer is full, then the producer shouldn't be

allowed to put any data into the buffer. The consumer should consume data only when the buffer is

not empty. If the buffer is empty, then the consumer

shouldn't be allowed to take any data from the buffer.

 The producer and consumer should not access the buffer at

the same time. What's the solution?

The above three problems can be solved with the help of

semaphores(learn more about semaphores from here). In the producer-consumer problem, we use three semaphore

variables:

1. Semaphore S: This semaphore variable is used to achieve

mutual exclusion between processes. By using this variable, either Producer or Consumer will be allowed to use or access

the shared buffer at a particular time. This variable is set to 1

initially. 2. Semaphore E: This semaphore variable is used to define

the empty space in the buffer. Initially, it is set to the whole

space of the buffer i.e. "n" because the buffer is initially

empty. 3. Semaphore F: This semaphore variable is used to define

the space that is filled by the producer. Initially, it is set to "0" because there is no space filled by the producer initially. By using the above three semaphore variables and by using

the wait() and signal() function, we can solve our

problem(the wait() function decreases the semaphore variable by 1

and the signal() function increases the semaphore variable by 1). So. let's see how. The following is the pseudo-code for the producer: void producer() {

```
while(T) {
produce()
wait(E)
wait(S)
append()

signal(S)
signal(F)
}
}
```

The above code can be summarized as:

 while() is used to produce data, again and again, if it wishes

to produce, again and again. produce() function is called to produce data by the producer. wait(E) will reduce the value of the semaphore variable "E" by one i.e. when the producer produces something then there

is a decrease in the value of the empty space in the buffer. If

the buffer is full i.e. the vale of the semaphore variable "E" is

"0", then the program will stop its execution and no

production will be done. wait(S) is used to set the semaphore variable "S" to "0" so

that no other process can enter into the critical section. append() function is used to append the newly produced data

in the buffer. signal(s) is used to set the semaphore variable "S" to "1" so

that other processes can come into the critical section now

because the production is done and the append operation is

also done. signal(F) is used to increase the semaphore variable "F" by

one because after adding the data into the buffer, one space is

filled in the buffer and the variable "F" must be updated. This is how we solve the produce part of the producer-consumer

problem. Now, let's see the consumer solution. The following is the

code for the consumer: void consumer() {

```
while(T) {
wait(F)
wait(S)
take()
signal(S)
signal(E)
use()
}
}
```

The above code can be summarized as:

 while() is used to consume data, again and again, if it wishes

to consume, again and again. wait(F) is used to decrease the semaphore variable "F" by one

because if some data is consumed by the consumer then the

variable "F" must be decreased by one. wait(S) is used to set the semaphore variable "S" to "0" so

that no other process can enter into the critical section. take() function is used to take the data from the buffer by the

consumer. signal(S) is used to set the semaphore variable "S" to "1" so

that other processes can come into the critical section now

because the consumption is done and the take operation is

also done. signal(E) is used to increase the semaphore variable "E" by

one because after taking the data from the buffer, one space is

freed from the buffer and the variable "E" must be increased. use() is a function that is used to use the data taken from the

buffer by the process to do some operation. So, this is how we can solve the producer-consumer problem.