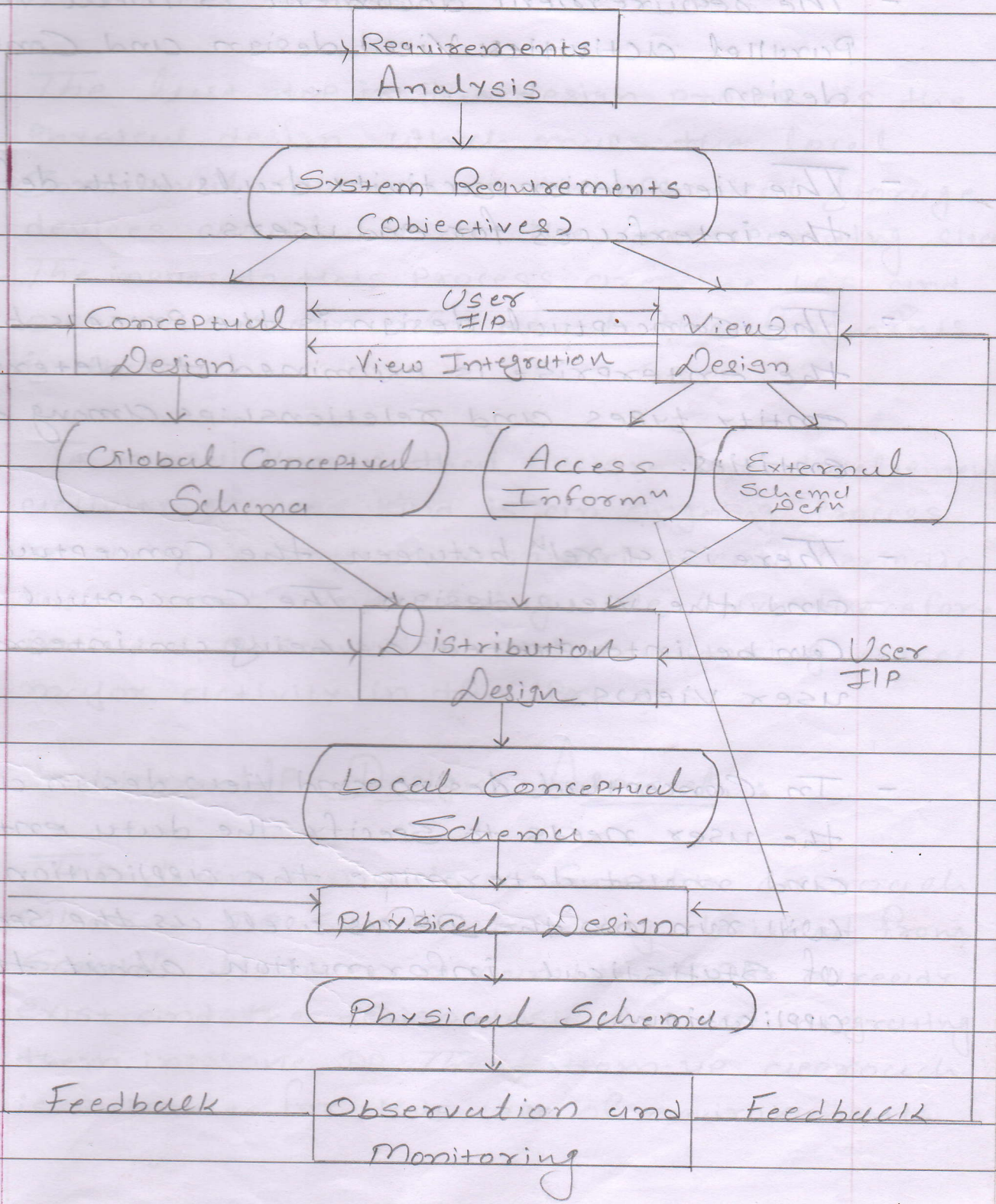
signature
DATE: / /

Ch: 4

Data Distribution Alternatives:

* Design Alternatives:

1) Top-down Approach:



signature

DATE: / /

- The activity begins with a requirement analysis the defines the environment of the System and Collect both the data and Processing needs of all potential database users."

- The requirement document is input to two Parallel activities: View design and Conceptual design.

- The view design activity deals with defining the interfaces for end users.

- The Conceptual design is the Process by which the enterprise is examined to determine entity types and relationships among these entities.

- There is a rel" between the Conceptual design and the view design. The Conceptual design can be interpreted as being an integration of user views.

- In Conceptual design and View design activities the user needs to Specify the data entities and must determine the application that will run on the DB as well as the Specification of Statistical information about these applications.

signature
DATE: / /

- The GCS and ~~also~~ access pattern information collected as a result of view design are ilp to the distribution design step. The objective is to design the local conceptual schemas by distributing the entities over the sites of the distributed system.

- The last step in the design process is the physical design, which maps the local conceptual schemas to the physical storage devices available at the corresponding site. The inputs to this process are the LCS and access pattern info. about the fragments in these.

- It is well known that design and development activity of any kind is an ongoing process requiring constant monitoring and periodic adjustment and tuning. We have therefore included observation and monitoring as a major activity in this process.

2) Bottom-Up Design Approach:

- Top-down design is a suitable approach when a DB system is being designed from scratch. However, a number of DBs already exist and the design task involves integrating them into one DB. The bottom-up approach is suitable for this type of environment.

signature

DATE : / /

:ψ ~~Distributed~~ Design Alternatives:

1) Localized data

2) Distributed data
   - ↳ No replicated, No fragmented
   - ↳ fully Replicated
   - ↳ fragmented or Partitioned
   - ↳ Partially Replicated
   - ↳ Mixed Distribution

* Fragmentation:

signature
DATE: / /

* View Management:

- A view is a virtual relation, defined as the result of a query on base relations, but not materialized like a base relation, which is stored in the DB.
- Dynamic window, reflects all updates to the Database.
- Ensure Data Security, hide some data, Select Subset of data.
- User access DB through views, they can't see or manipulate the hidden data, which are therefore Secure.

:w) Views in Centralized DBMSs:

Ex: EMP(ENO, ENAME, TITLE)

```
Create View SYSANL (ENO, ENAME)
AS Select ENO, EName
    From EMP
    Where Title = "Syst. Anal."
```

signature

DATE : / /

Updates through Views:

- Views can be defined using Selection, join, Projection, aggregate fun. and so on.
- View is updatable only if the updates to the view can be propogated to the base rel^n without ambiguity.
- The View SYSAN is updatable; the insertion ex: <201, Smith>
- following is not updatable:

```
Create View EG(ENAME, RESP)
AS SELECT ENAME, RESP
     from EMP, ASG
     WHERE EMP.ENO = ASG.ENO
```

- Views can be updated only if they are derived from a Single relation by selection and Projection.

Views in Distributed DBMSs:

Views in

- Distributed System may be derived from fragmented rel^n. Stored at different Sites.
- When a view is defined, its name and its retrival query are stored in the Catalog.

signature
DATE: / /

- Since views may be used as base relations by application programs, their definition should be stored in the directory in the same way as the base relation descriptions.

- Depending on the degree of site autonomy offered by the system, view definition can be centralized at one site, partially duplicated or fully duplicated.

- In any case, the info. associating a view name to its definition site should be duplicated. If the view definition is not present at the site where the query is issued remote access to the view definition site is necessary.

- Views derived from distributed relations may be costly to evaluate. An alternative soln is to avoid view derivation by maintaining actual versions of the views, called Snapshots.

- Snapshots ↳ represent particular state of DB.
  ↳ Static, does not reflect update to base relations.
  ↳ Useful when user are not intrested in seeing the most recent version of the DB.
  ↳ Necessary to recalculate Snapshots periodically., done when system is idle.

signature

DATE : / /

* Semantic Integrity Control :

- DB Consistency, if DB Satisfies a set of Constraints, called Semantic Integrity Constraints
- Semantic Integrity Control ensures DB Consistency by rejecting update programs which lead to inconsistant DB State.

:\ Centralized S. I. C. :

→ Specification of integrity Constraints:

↳ These Constraints can be defined either at rel^n creation time or at any time, even if the rel^n already contains tuples.

↳ In RDBMS, integrity Constraints are defined as assertions.

↳ Three types of integrity Constraints.

① Predefined - based on Simple Keyword.
Ex: NOT NULL, Unique, foreign key

② Precompiled - Express Preconditions that must be Satisfied by all tuples in a rel^n for a given update type
→ Update type - INSERT, DELETE or MODIFY
Ex:
- CHECK ON PROJ (BUDGET ≥ 50K AND BUDGET ≤ 100K)
- CHECK ON PROJ WHEN DELETE (BUDGET = 0)

③ General Constraint: are formulas of tuple relational Calculus where all variable are quantified. DB must ensure that those formulas are always true.

signature

DATE: / /

Ex: CHECK ON e1:EMP, e2:EMP
(e1.ENAME = e2.ENAME IF e1.ENO = e2.ENO)

→ Integrity Enforcement:

- Consist of rejecting update programs that violate some integrity constraints.
- Two methods:

① Detection:
- Update U is executed, change DB state D to Du.
- Apply test derived from these constraints.
- If state Du is inconsistant, Restore D by undoing U.
- Called Posttests.

② Prevention
- An update is executed only if it changes the DB state to a consistant state.
- Pretest.
- More efficient than Posttest since update never need to be undone because of integrity violation. Ex is Query modification algo.

Ex: UPDATE PROJ SET BUDGET = BUDGET * 1.1
WHERE PNAME = "CAD/CAM"

↓

UPDATE PROJ. SET BUDGET = BUDGET * 1.1
WHERE PNAME = "CAD/CAM" AND BUDGET ≥ 50K
AND BUDGET ≤ 100K
New. New.

signature

DATE : / /

Distributed Semantic Integrity Control :

- Two main problems of designing an integrity subsystem for a DDBMS are the definition and storage of assertions, and the enforcement of these assertions.

→ Definition of Distributed Integrity Assertions :

- Since assertions can involve data stored at different sites, their storage must be decided so as to minimize the cost of integrity checking.

- Three classes of assertions :

1. Individual assertions : Single-relⁿ Single variable assertions.
   [Domain Constraint]
   Ex : CHECK ON PROJ (BUDGET ≥ 50K AND BUDGET ≤ 100K)

2. Set Oriented assertions : Single relation - multivariable Constraints
   Ex : functional Dependency :→
   ENO IN EMP DETERMINES ENAME
   - foreign Key :→
   PNO IN ASG REFERENCES PNO IN PROJ

3. Assertions involving aggregates : Require Spacial Processing because of the cost of evaluating the aggregates.

signature

DATE: / /

→ Individual assertions:

Assertion definition is sent to all other sites that contain fragments of the rel" involved in the assertion.

- Compatibility is checked at two levels: Predicate and data
- first, Predicate Compatibility is verified by Comparing the assertion Predicate with the fragment Predicate. If noncompatibility is found at one of the sites, the assertion definition is globally rejected because tuples of that fragment do not satisfy the integrity constraints.
- If Predicate Compatibility has been found, the assertion is tested against the instance of the fragment. If not satisfied by the instance, the assertion is globally rejected.
- If Compatibility is found, the assertion is stored at each site.

Ex.: EMP, horizontally fragmented across three sites using Predicates:

P1: ENO < "E3"
P2: "E3" ≤ ENO ≤ "E6"
P3: ENO > "E6"

- Domain assertion C: ENO < "E4".

C is Compatible with P1 and P2, but not with P3, so C is globally rejected.

signature

DATE: / /

→ Set-Oriented assertions :

- Are multivariable, Involve join Predicates
- Assertion def" Can be Sent to all the sites that Store a fragment referenced by these Variables.
- Predicate Compatibility is useless (based on a join predicate)
- C must be Checked for Compatibility against the data.

→ Enforcement of Distributed Integrity Assertions :

- More Complex than Centralized DBMS.
- Problem is to decide where to enforce the integrity assertions.
- Choise depends on the class of the assertion type of update and the nature of the site where the update is issued.
- Site may, or may not, store the updated rel" or Some of the rel" involved in the integrity assertions.

(1) Individual Assertions :

- Two Case are Considered.
- Update is an insert Statement, individual assertions Can be enforced at the site where the update is Submitted.

Signature
DATE: / /

- If update is delete or modify statements, it is sent to the sites storing the rel^n that will be updated.

② Set-Oriented assertions:

- Site where the update is submitted must receive for each site a message indicating that this assertion is satisfied and that it is a condition for all sites.
- If assertion is not true for one site, this site sends and error message indicating that the assertion has been violated, the update is then invalid.

③ Assertion involving aggregates:

- Most costly among all, require calculation of the aggregate functions.
- MIN, MAX, SUM & COUNT, each contain Projection and Selection part.