Date: DBMS Day: M T W T F S

Ch: 7

~~Data modelling~~ XML & JSON used for data interchange

⇒ Extensible Markup language

⇒ Tag : label for a section e.g <header>

⇒ Element : Data Section beg. <tagname> end </tag>

⇒ Attributes: Represented by "name = value" inside opening tag of an element

Attributes vs Subelements

⇒ Subelement : <tag1> <tag2> ... </tag2> </tag1>

⇒ Attr. are part of ~~document~~ markup

⇒ Subelement are part of basic document

⇒ Suggestion : Attr → id of elements
sub-ele → contents

⇒ String that used as tags
<![CDATA[<account> ... </account>]]>
Here <account> and </account> are treated as strings

⇒ Document Schema
⇒ DTD (Document Type Definition)
⇒ XML Schema (New)

Duplicate

Date:

Day: M T W T F S

⇒ JSON

⇒ String, number, object, array, ~~true, false~~ bool, null

⇒ Can be used by JSON function.

Ch: 11

NO SQL

Why

⇒ Joins are expensive

⇒ Hard to scale

⇒ Impedance mismatch

⇒ Expensive product cost

⇒ Speed

⇒ Partition

⇒ Availability.

⇒ Characteristics

Avoids

⇒ Avoids overhead Acid transactions

⇒ Complexity of SQL

⇒ Schema design

⇒ DBA presence

Provides

⇒ Provides easy and fast changes to DB

⇒ Fast dev

⇒ Large data (Big data e.g. Google)

⇒ Schemaless

Date: Day: M T W T F S

When/Why?

- RDBMS is restrictive (Flexible schema needed)
- ACID not "really" needed
- Logging data from distributed systems
- Temporary Data (Carts, sessions, favorites)
- Polyglot Persistance:
  Best Data store depending on data

Schema less

- In RDB
  - Can't add record with different datatypes, less attr., or multiple ~~fields~~ data.
  - Should consider Primary key, Joins etc
- In No-SQL
  - No Schema, no-unneeded, no-datatype

Aggr. Data Models

- Key-value (redis) → Document (mongoDB)
- Col. family (Apache) → Graph (Neo4j)

- **Key-Value**
  - Easiest, access data using keys (string, hashed)
  - Operations: Insert, Fetch, Update, delete
- **Col. family**
  Cassandra → facebook Search

Date: Day: M T W T F S

⇒ MySQL > 50GB
  ⇒ Write : ~ 300ms, Read: ~ 350ms
⇒ Cassandra > 50GB
  ⇒ write: 0.12 ms, read: 15ms

⇒ Document

⇒ Pair key with complex data structure
⇒ Indexes with B-tree
⇒ Nested documents

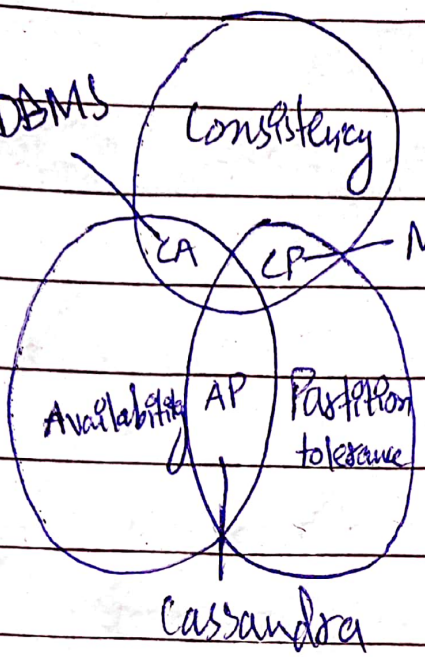
SQL vs NO-SQL (Diff)

⇒ mySQL vs mongo
⇒ Row & tables vs key-value, JSON, XML
⇒ Schemas: Static vs Dynamic
⇒ Scaling: vertical, horizontal vs horizontal
⇒ Data Manipulation: Queries vs OO API's

CAP theorem:

⇒ Scalability, Consistency, availability, Fault-tolerance
  Impossible at a time.
⇒ Impossible for any shared data-system
to guarantee simultaneously all three
→ consistency → Availability → Partition tolerance

Date: Day: M T W T F S



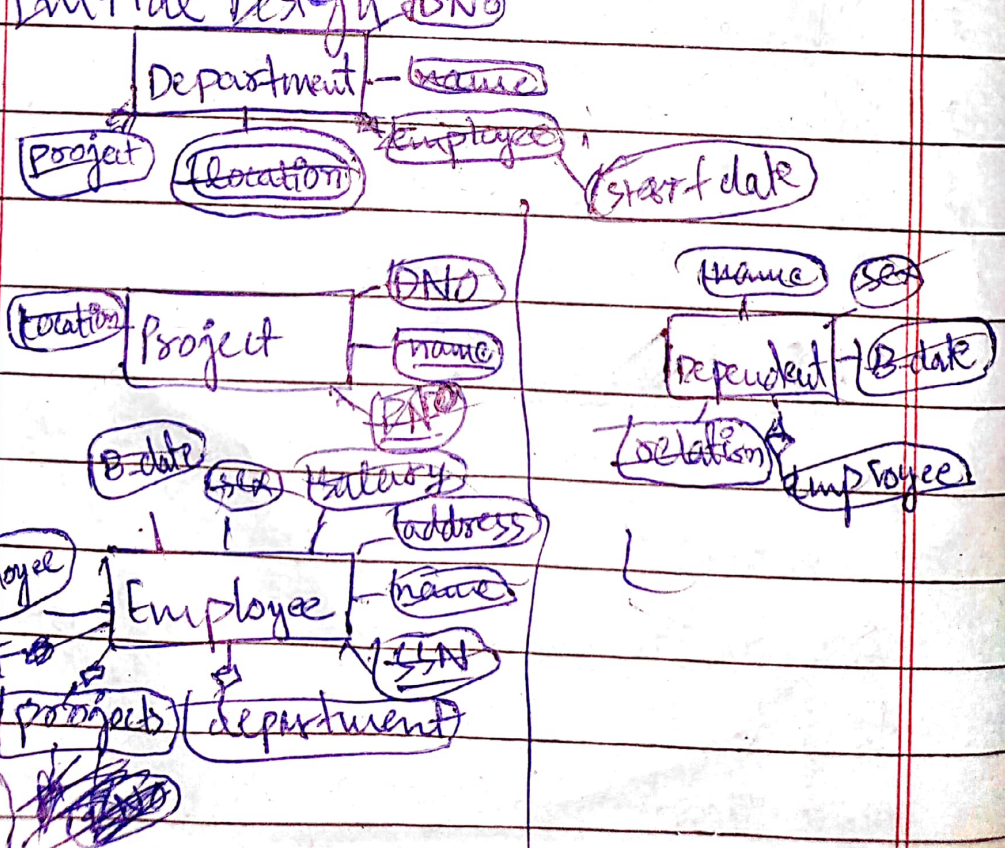
NOSQL : AP

RDBMS : CP

Ch:7

Company (ER)

2) Initial Design

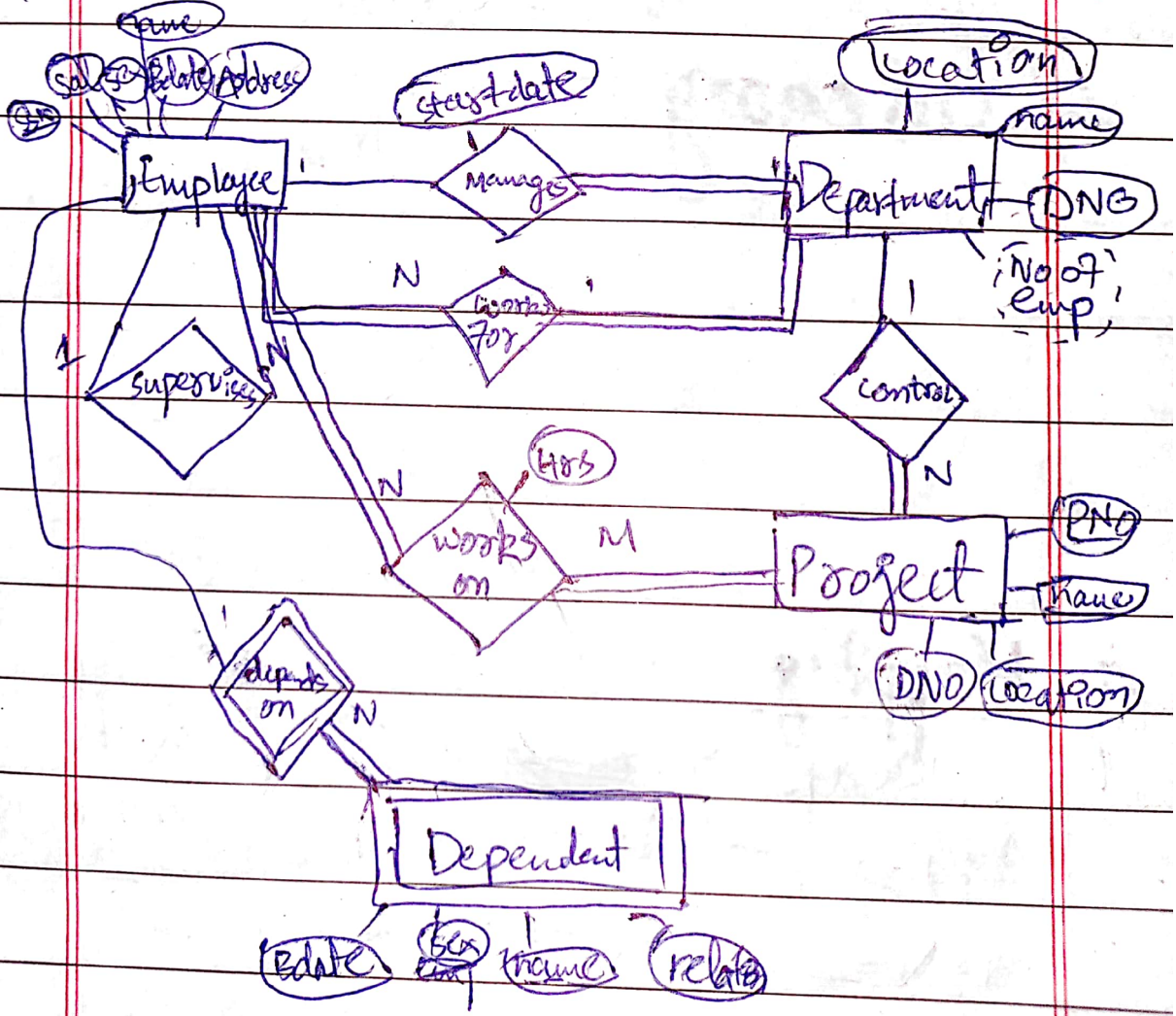


Date: Day: M T W T F S

⇒ Relationship types

- ✓ ⇒ manages emp → dep
- ✓ ⇒ works on emp → proj
- ✓ ⇒ depend on emp → depend
- ✓ ⇒ works for emp → dep
- ✓ ⇒ Supervises emp → emp
- ✓ ⇒ Controls dep → proj

⇒ ER



XML to JSON

Date: Day: M T W T F S

⇒ Cardinality (max 1:M)

⇒ Modality (zero, one or more) min

⇒ **High Degree Relations**

⇒ Uni me kar ga

⇒ **EER Class**

⇒ d must use

⇒ no or o means optional

⇒ refinement ⇒ spec

synthesis ⇒ gen

⇒ **Category** (union type)

⇒ **Mapping**

weak ⇒ include PK of strong

1:1 ⇒ PK → FK

1:N ⇒ 1 PK → N FK

M:N ⇒ Make new table and include both PK

Multivalued ⇒ New table include PK and attr.

Date: Day: M T W T F S

Ch#15

Normalization

⇒ Informal Design

⇒ ~~Semantics~~

⇒ Bottom up ⇒ adding attr to form relations.

⇒ Top down ⇒ starts group of attr to form relations

⇒ Guidelines

- semantics
- Redundancy
- null
- spurious

⇒ 1st Guide:

⇒ each tuple should represent one entity or relationship

⇒ Attr. of diff relation shouldn't mixed.

⇒ Only FK for referencing

⇒ Entity and relationship should be kept apart.

⇒ 2nd Guide

⇒ Mixing attr. that depends of other

⇒ Anomalies (Insert, delete, update)

⇒ Use views and update them by toggers.

Date: Day: M T W T F S

13rd Guide:

- Separate table with PK
- Reason (not applicable, invalid, unknown, unavailable)

4th Guide

- Bad Design may result in erroneous results for JOIN, use lossless JOIN
- Relations should be designed for lossless JOIN and doesn't spurious tuples
- There are 2 imp decomp. prop.
  - lossless or non-additive
  - Preservation of FD

spurious

FD: Constraint b/w 2 attr interrel.

- Used to define NF
- A set of attr X functionally determines a set of Y if X determines unique Y
- $X \rightarrow Y$, 2 tuples same X must have same Y

Inference Rules (Armstrong)

- IR_1 (reflexive): Y subset-of X : $X \rightarrow Y$
- IR_2 (Augmentation): If $X \rightarrow Y$ then $XZ \rightarrow YZ$
- IR_3 (Transitive): If $X \rightarrow Y$: $X \rightarrow Z$: $Y \rightarrow Z$ then

Date: Day: M T W T F S

Additional

⇒ Decomposition: $X \to YZ$: $X \to Y$, $X \to Z$

⇒ Union: $X \to Y$ and $X \to Z$ then $X \to YZ$

⇒ Pseudotransitivity: $X \to Y$, $WY \to Z$ then $WX \to Z$

⇒ Normal forms

⇒ Normalization: process of decomp. bad relations by breaking into multiple.

⇒ 1, 2, 3 FDs, 4 MVD, 5, JD

⇒ Denormalization (ulta krdo)

⇒ Prime attr are part of candidate key.

⇒ 1st NF:

⇒ Atomic values

⇒ ~~composite~~, ~~multivalued~~, ~~nested rel~~

⇒ Make multiple tuples

⇒ 2nd NF

⇒ Non primes (PnR) are fully FD on PK