DBMS

Page No.:
Date:
YOUVA

M T W T F S S

base → collection of interrelated data
→ Entities & Relationships
convenient and efficient access of data

tion: External / View level → how users see
Logical / Conceptual level → what data (column names & attributes) & relationship among them
Physical level → how data actually stored (eg unordered map order as primary key)
changes in one doesn't affect others

cify the database schem

& updates

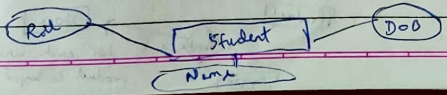
in table
attributes rows
in table
column types / primary key / foreign key

:

the table (has a set of attributes)
course, a student (distinguishable objects)

entire table (collection of entities having same property)
eg: (courses, Students entity set

Attributes → column names, eg: SName, Roll, Gender



Database Theory

DBMS

M T W T F S S
Page No.:
Date:
YOUVA

* Database → collection of interrelated data
  Model → Entities & Relationships
  DBMS → convenient and efficient access of data

* Levels of Abstraction: External / View level → how users see
  Logical / Conceptual level → what data (column names & attributes) & relationship among them
  Physical level → how data actually stored (eg unordered map index as primary key)

↳ All levels are independent: changes in one doesn't affect others

* Database system:

i) DDL (Data Definition language) → specify the database schema
   eg: create table,

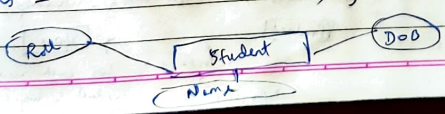
ii) DML (data manipulation language) → queries & updates
   eg: select → retrieve
   insert → add fields (rows) in table
   delete → delete ~~column~~ / ~~attributes~~ rows
   update → modify values in table
   alter → modify column types / primary key / foreign key

* Entity Relationship Model:

↳ Entity: a row of the table (has a set of attributes)
   eg: a course, a student (distinguishable objects)

↳ Entity set → entire table (collection of entities having same property)
   eg: (course, Student) entity set

↳ Attributes → column names, eg: SName, Roll, Gender



* Types of Attributes.

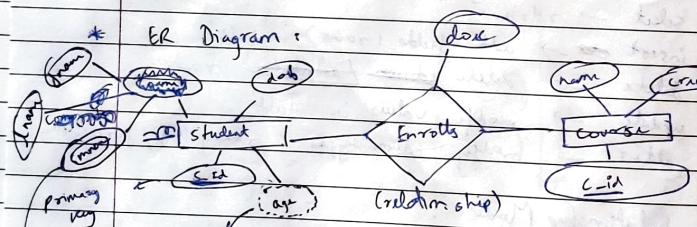
↳ Simple: indivisible eg dept. / phone no.
↳ Composite: several components eg: Qualification → Btech, 2019, IIT
name → first, middle, last
↳ Derived: dependent on some other attribute eg: age on DOB

+ Types of Attributes (2):

↳ Single valued → have only one value eg: Place of Birth
↳ Multivalued → have set of values eg: Courses enrolled, email
↳ combination of type 1 & 2

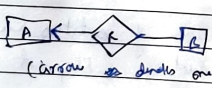
* Keys: uniquely identifies an entity
↳ can be a combination of attributes
↳ entity set can have multiple keys

* ER Diagram:



Key enrolls = { S_id, C_id }

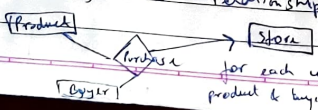
* One to one, one to many, many to one, many to many



(arrow → double one) : one to many
↳ each A can have multiple B
↳ each B can have atmost one A

Degree = 2 (no. of participating entities in a relationship)

Ternary eg:



for each unique combination of product & buyer, atmost one store associated

A	B
a_1	b_1
a_2	b_2

r

B	C
b_2	c_2
b_3	c_3

s

outer join → null values added

$r \bowtie s =$

A	B	C
a_2	b_2	c_3
a_1	b_1	null
null	b_3	c_3

→ included in left outer join

→ included in right outer join

Left outer join

$\bowtie \rightarrow r \bowtie s \cup (r - \Pi_R(r \bowtie s)) \times \{(null, \ldots, null)\}$

a_1, b_1, null

A	B	C
a_2	b_2	c_3

Full outer join $\bowtie$

Division :

$R = \{a_1, a_2 \ldots a_n, b_1, b_2, \ldots b_n\}$ ← attributes

$S = \{b_1, b_2, \ldots, b_n\}$

$r \div s \rightarrow$ on $R - S$

r

A	B
α	1 ✓
α	2 ✓
α	3
β	1 ✓
γ	1
δ	1
δ	3
δ	4
ϵ	6
ϵ	1
β	2 ✓

s

B
1
2

$r \div s =$

A ∈ (R−S)
α
β

For γ, δ, ϵ both 1 & 2 are not present

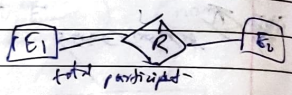
M T W T F S S
Page No.:
Date: YOUVA

Min - max notation :

E_1 — (m,n) — R — (m,n) — E_2

m : minm no. of times a particular entity must appear in relationship tuples

0 → partial participation
1/1 → total participation

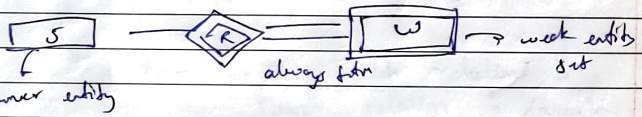


n : maxm. ——

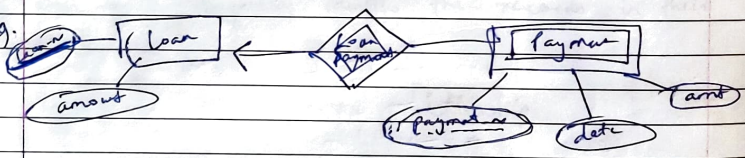
Weak Entity Set :

⇒ no primary key

→ each weak entity is associated with some entity of owner entity set.



eg.



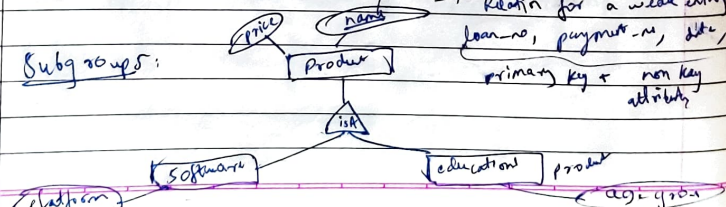
L_1 25000	20	24/7/21	5000
L_2 30000	20	29/7/21	6000
L_3 48000	20	24/7/21	5000

on some date payment made of 5000 each

can't determine associated with which loan_no

→ Relation for a weak entity will primary key + non key attributes

Subgroups :



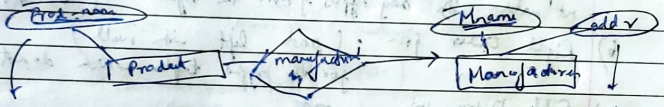
Entity set vs Attributes:

* To be an entity set, one of the two conditions must satisfy.

i) has atleast one non-key attribute

ii) it is the many in many-one (many-many relationship)

eg:



good design

many side (many prod. can have same manufacturers)

has non key attribute

Categories of language

Procedural
→ Relational algebra

Non-procedural
→ Tuple relational calculus
→ Domain relational calculus

Relational Algebra:

Basic operators:

Selection (σ), Projection (π), Union ($\cup$), set difference ($-$), cartesian product ($\times$), Renam: ρ

↳ $\rho_x(E)$: returns expression E under the name x

Additional Operators:

i) Set intersection ($\cap$) : $r \cap s = r - (r - s)$

ii) Natural join : $r \bowtie s$

↳ cartesian product + equating common attributes + removing duplicates

$$r \bowtie s = \prod_{R \cup S} \left(\sigma_{r.A_1 = s.A_1 \wedge r.A_2 = s.A_2 \wedge \dots r.A_n = s.A_n} (r \times s) \right)$$

iii) Outer join operation:
natural join + add tuples which do not match using null values

Types:
a) Left Outer join: includes tuples from left relation that did not match with any tuples from the right. ($r \bowtie s$)
b) Right Outer join: (pads left with null) for the right tuples which do not have match
c) Full Outer join: $\bowtie$

iv) Division Operator: (used when for all)

$$r \div s = \{ t \mid t \in \Pi_{R-S}(r) \wedge \forall u \in s, (tu \in r) \}$$

$$= \Pi_{R-S}(r) - \Pi_{R-S}\left((\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r) \right)$$

Assignment Operator:
eg $r \bowtie s$: $temp1 \leftarrow r \times s$
$temp2 \leftarrow \sigma_{r.A_1 = s.A_1 \ldots r.A_m = s.A_m}(temp1)$
$result \leftarrow \Pi_{R \cup S}(temp2)$

few more join operation
- Semi join:
  r semiJoin $s = \Pi_R(r \bowtie s)$
- Anti join:
  r antijoin $s = r - (r$ semiJoin $s)$

Delete : $r \leftarrow r - E$ ($r \rightarrow$ relation, $E \rightarrow$ Retrieved algebra query)

Insert : $r \leftarrow r \cup E$
Update : $r \leftarrow \Pi_{F_1, F_2, \ldots F_i}(r)$

Example :

Find the names of all customers who have a loan at 'Patalipatra' branch but do not have an account at any branch of the bank.

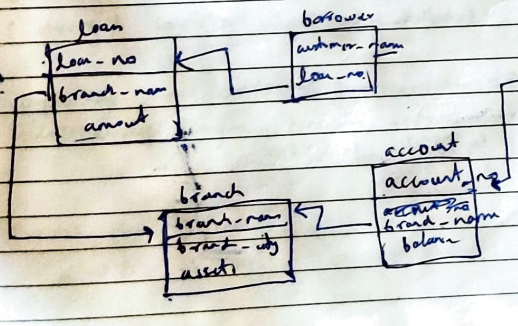
Relational algebra : $\Pi_{customer\_name}(\sigma_{branch\_name = 'patalipatra'}(borrower \bowtie loan)) - \Pi_{customer\_name}(depositor)$

Tuple Relational Calculus (allows) : $\{t \mid \exists s \in borrower (t[customer\_name] = s[customer\_name] \wedge \exists u \in loan (u[branch\_name] = "Patalipatra" \wedge u[loan\_no] = s[loan\_no])) \wedge \text{not } \exists v \in depositor (v[customer\_name] = t[customer\_name])\}$

$t \rightarrow$ for desired attribute

$s, u, v \ldots \rightarrow$ for denoting tuples in each relation

Domain Relational Calculus : $\{ \langle c \rangle \mid \exists l, b, a (\langle c, l \rangle \in borrower \wedge \langle l, b, a \rangle \in loan \wedge b = "patalipatra") \wedge \text{not } \exists ac (\langle c, ac \rangle \in depositor)\}$



M T W T F S S
Page No.:
Date:
YOUVA

Q) Find the names of customers who have an account at all the branches located in 'Patna' city.

Relational:

$$\Pi_{customer\_name,\ brand\_name}(depositor \bowtie account) \div \Pi_{branch\_name}(\sigma_{branch\_city = \text{"Patna"}}(branch))$$

TRC:

$$\{t \mid \forall u \in branch\ (u[branch\_city] = \text{"Patna"} \Rightarrow \exists v \in account\ (v[branch\_name] = u[branch\_name] \wedge \exists w \in depositor\ (w[account\_no] = v[account\_no] \wedge (t[customer\_name] = w[customer\_name])))\}$$

DRC

$$\{\langle c \rangle \mid \forall x,y,z\ (\langle x,y,z \rangle \in branch \wedge y = \text{"Patna"}) \Rightarrow \exists a,b\ (\langle a,x,b \rangle \in account \wedge \langle c,a \rangle \in depositor)\}$$

Q) Find the largest account balance in the bank

$$\Pi_{balance}(account) - \Pi_{account.balance}(\sigma_{account.balance < d.balance}(account \times \rho_d\ account))$$

Functional Dependency & Normalization

→ loss less join decomposition:
↳ decomposing a table into smaller units: natural join will give original table

1st Normal Form:
→ no multivalued / composite attributes
* A relational schema R is in 1st Normal form if domain of all attributes of R are single atomic values.

Functional Dependency:
→ values of certain set of attributes determine uniquely the value for another set of attributes (like a Key)

⇒ $\alpha \rightarrow \beta$
$\forall\, t_1, t_2 \in r \;\; \{(t_1[\alpha] = t_2[\alpha]) \Rightarrow (t_1[\beta] = t_2[\beta])\}$
(if value repeats in α, then it must repeat in β)

A	B
1	4
1	5
3	7

$A \rightarrow B$ ✗
$B \rightarrow A$ ✓

⇒ K is superkey for R iff $K \rightarrow R$

⇒ K is candidate key for R iff $K \rightarrow R$ and for no $\alpha \subset K$, $\alpha \rightarrow R$
(strongest condition → least set of attributes)

⇒ Super Keys → constraint for entire table / relation R
Functional dependency → can also express other constraints (set of attributes)

* Trivial Func. Depend.:
$\alpha \rightarrow \beta$ is trivial if $\beta \subseteq \alpha$

* Closure of $F \rightarrow F^+$:
set of all func. dependency logically implied by F

* Armstrong's Axioms :
→ if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$ (reflexivity)
→ if $\alpha \rightarrow \beta$ then $\gamma\alpha \rightarrow \gamma\beta$ (augmentation)
→ if $\alpha \rightarrow \beta$ and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$ (transitivity)

* Additional rules :
→ Union : if $\alpha \rightarrow \beta$ and $\alpha \rightarrow \gamma$, then $\alpha \rightarrow \beta\gamma$
→ decomposition : If $\alpha \rightarrow \beta\gamma$, then $\alpha \rightarrow \beta$ and $\alpha \rightarrow \gamma$
→ pseudotransitivity : if $\alpha \rightarrow \beta$ and $\gamma\beta \rightarrow \delta$ then $\alpha\gamma \rightarrow \delta$

Full functional Dependency :
→ minimal in size
→ FD $X \rightarrow A$ for which there exist no proper subset Y of X : $Y \rightarrow A$, then A is fully functionally dependent on X.

Closure of Attribute Sets :
→ set of all attributes functionally determined by a.
→ denoted by a^+
$A^+ = \{A, B, C, D\}$ means we can uniquely determine A, B, C, D when we know A.

Cover of a set of FDs :
→ F is a cover of G or vice versa if $F^+ = G^+$ where F and G be set of functional dependencies

* Minimal Cover or canonical cover (F_c) :
↳ no redundant terms

Extraneous Attribute:

→ attribute which can be removed without changing the closure of a set of FDs.

→ Let $\alpha \to \beta$ be in F

A is extraneous in α if A ∈ α and F logically implies $(F - \{\alpha \to \beta\}) \cup \{(\alpha - A) \to \beta\}$

→ A is extraneous in β, if A ∈ β and $(F - \{\alpha \to \beta\}) \cup \{\alpha \to (\beta - A)\}$ logically implies F.

2NF : 2nd Normal form:

→ R is in 2NF if every non-prime attribute A in R is not partially dependent on any candidate key of R. (Prime attribute → attribute part of candidate key)

→ eg: r(A, B, C, D)

if (A, B) is a candidate key & A → D holds

Then decompose into r_1 & r_2

r_1(A, D) → A is the candidate key

r_2(A, B, C) → {A, B} is candidate key

2NF
eg: R(A, B, C, D)
A, B → D
B → C
Non prime → C, D
(AB) → candidate key

3NF:

→ For a non-trivial FD X → A to hold either

i) X is a superkey of R or

ii) A is a prime attribute of R

B → C unprime (partial dependency)
R1(A, B, D)
R2(B, C)

→ if r(A, B, C) and A → candidate key & B → C holds

Then r can be replaced by

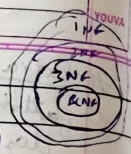
r_1(B, C) → B is the candidate key

r_2(A, B) → A is the candidate key

BCNF:

→ 3NF with only the first condition
($X \to A$ holds then X is a superkey of R)

→ stronger as every ~~3NF is BCNF~~ BCNF is 3NF



* For 2NF it should be in 1 NF
and for 3NF it should be in 2NF.

eg: 3NF: R(ABCD)
Candidate key : AB
2NF { i) $AB \to C$ — E,D → non prime
ii) $C \to D$ → transitivity (not allowed)
non prim → non prime (allowed in 2NF)

R_1 (A, B, C) (In 3NF NP → NP not allowed)
R_2 (C, D)

eg BCNF

R (A B C)

$AB \to C$
$C \to B$
B not a superkey

Lossless Join :

- Let R be relation schema & F be set of func. dependency
Let R_1, R_2 are decomposition of R
- The decomposition is lossless if atleast one of the two func. dependencies is in F^+ :

i) $R_1 \cap R_2 \to R_1$
ii) $R_1 \cap R_2 \to R_2$

$R_1 \cap R_2$ = common attribute = B (C) superkey of R_1 or R_2 B is / is not superkey

eg $r\langle A, B, C \rangle$
candidate key = C and $A \to B$

r_1 (A, B) — $(A \to B)$
r_2 (B, C) — $(C \to B)$
(not lossless join)

Dependency preservation:

same eg:

$r(A,B,C)$

$r_1(A,B)$: $A \to B$

$r_2(B,C)$: $C \to B$

$C \to A$

$C \to B$ ✓

$A \to B$ ✓

Can we derive $C \to A$ from $A \to B$ and $C \to B$

→ not possible to obtain ⇒ not dependency preserving

eg 2) Given: $A \to B$, $B \to C$

⇒ $A \to C$

$r(A,B,C)$

$r_1(A,B)$: $A \to B$

$r_2(A,C)$: $A \to C$

⇒ not dependency preserving

can't derive $B \to C$

Transactions

M T W T F S S
Page No.:
Date: YOUVA

ACID Properties :
To preserve the integrity of a database

i) Atomicity : either all operations or none are reflected in database

ii) Consistency : explicit like primary keys, foreign keys
implicit like sum (A+B) after transaction = sum (A+B) before transaction

iii) Isolation : Each transaction unaware of other concurrently executed transactions
→ can be ensured trivially by running transactions serially (one transaction completed before executing other)

iv) Durability : Database updates after transaction must persist even after software / hardware failures

Concurrency control schemes
→ mechanisms to achieve isolation i.e control interactions b/w concurrent transactions to ensure consistency

Schedule : sequence of instructions in chronological order in which instructions of concurrent transactions are executed

→ all instructions must be present & order of instructions in a transaction must be preserved.

→ A schedule is serializable if it is equivalent to a serial schedule

Assumption → each transaction preserving consistency
⇒ serial execution preserves consistency

Conflict Serializability:

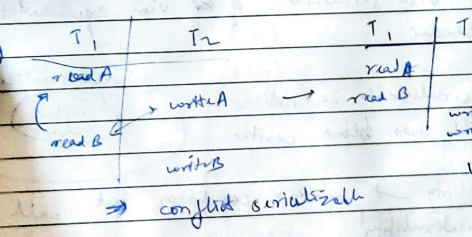
→ Instructions I_i & I_j of transaction T_i & T_j conflict iff some item Q is accessed by both T_i & T_j & atleast one of them wrote(Q).

→ If no conflict b/w I_i & I_j then they can be swapped in schedule.

→ S $\xrightarrow{\text{swaps of non conflicting ins.}}$ S', then S & S' are conflict equivalent.

→ S is conflict serializable if it is conflict equivalent to a serial schedule

e.g.



⇒ conflict serializable

→ Precedence graph: directed graph : vertices → transaction
$T_i \rightarrow T_j$ if T_i executes before T_j & both are conflicting

→ A schedule is conflict serializable iff its precedence graph is acyclic. To find the serial order - use topological sort (can be multiple ordering)

View Serializability.

→ S and S' are view equivalent if for each data item Q:

i) In S Ti reads initial value of Q ⇒ In S' also Ti reads Q initial.

ii) In S Ti reads Q after Tj (write Q) ⇒ In S' also Ti read(Q) after Tj write(Q)

iii) In S if Ti writes final, then S' also same.

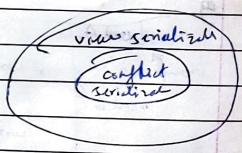
→ eg:

T_3	T_4	T_6
read(Q)		
	write(Q)	
write(Q)		
		write(Q)

$\sim (T_3, T_4, T_6)$

→ Schedule is view serializable if view equivalent to a serial schedule.

→ Every view serializable i.e not conflict serializable has blind writes.



→ View serializability → weaker condition but will ensure (serializability).

→ Even if not serializable, a schedule can produce same results.

Recoverable Schedule :

→ If Tj reads a data item previously written by Ti, then commit of Ti appears before the commit of Tj. (If Ti fails/abort, Tj must also abort)

Cascadeless schedule :

→ T_j reads a data item previously written by T_i, then commit of T_i must appear before read of T_j.

→ every cascadeless schedule is also recoverable.

Concurrency Control :

→ Schedules should be either conflict / view serializable & recoverable and preferably cascadeless.

1b) C

c) S2 :

T_1	T_2
	R(x)
	R(y)
	W(y)

$T_2 \rightarrow T_1$

S3 : $T_1 \rightarrow T_2$ ✓

2) $T_1 \rightarrow$ ⊗ younger

$T_2 \rightarrow$ delay

STORAGE

Page No.: Date: YOUVA

→ Cache > RAM > Flash Memory > Magnetic Disk > Optical Disk > Magnetic tape

primary memory — secondary — Tertiary

Magnetic Disk :

→ Read write head → magnetically reads / writes
→ tracks (circular)
→ sectors : smallest unit of data that can be read / written
~ 512 bytes

→ Request for disk I/O specifies the address on the disk to be referenced in the form of block no.
→ Block : Consist of fixed no. of contiguous sectors
→ Data transferred from disk ⟷ main memory in units of blocks

* Access Time = seek time + rotational latency time
(positioning the arm under correct track) (waiting for the desired sector to appear under R/W head)
→ time b/w a R/W request and beginning of data transfer.

* Data Transfer Rate :
→ rate at which data can be retrieved from or stored to the disk. (after R/W head positions on the first sector)

* Mean Time to Failure : reliability of disk
amount of time the system runs continuously without any failure.

RAID : Redundant Array of Independent Disks.
→ multiple disks / one view / higher speed & reliability
→ Redundancy to rebuild info lost in a disk failure.
eg: Mirroring → duplicate every disk

Raid 0 — block striping
non-redundant striping

Raid 1 — mirrored disk

Raid 2 — memory style error correction code (bit striping)

Raid 3 — bit interleaved parity

Raid 4 — block " "

Raid 5 — block " distributed parity

Raid 6 — P + Q redundancy
(extra redundant info)

FILE ORGANISATION :

→ each record entirely in one block
→ Block may have several records

- Fixed length Records
  - → free lists
- Variable length records:
  - → Byte String Representation } Variable length representation
  - → Slotted page structure
  - → Reserved space method
  - → list representation
  - → alternate list representation } Fixed length representation

M T W T F S
Page No.:
Date:
YOUVA

Organisation of records in FILES:
→ Heap (anywhere)
→ Sequential (based on search key)
→ Hashing (.. f(attribute) = block address of record)

• Generally separate file for each relation - except multitable clustering file organisation

Sequential File Org^n:
→ sorted order based on search key
→ records linked together by pointers
→ records stored physically in search key order or as close as possible

Multitable clustering file Organisation:
→ several relations in 1 file
→ good for queries of ⋈ (bad for only 1 table queries)

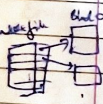
* Data Dictionary storage — stores metadata

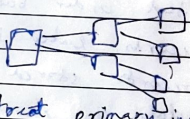
Indexing and Hashing:
→ Index file [search key | pointer]
→ Types of Indices:
  → Ordered indices : search key stored in sorted order
  → Hash indices : " " distributed uniformly across buckets using hash fn

→ Ordered indices (the sequential order of file)
  → Primary index: index whose search key specifies the sequential order of file
  → Secondary index: " " " " order diff. from sequential order of file

Index Sequential File : sequential file with a primary index
→ Dense (index record for every srch key value in file)
→ Sparse (contains index records for some search key values in file)

Sparse Index Files :
→ good tradeoff : sparse index with an index entry for every block in file corresponding to least srch key value in file



Multilevel Index :


→ When primary index file too big, treat primary index kept on disk as a sequential file & construct a sparse index on it.
→ outer index : sparse index of primary index inner index : primary index file

Secondary Index :
→ have to be dense
→ index record points to a bucket that contains pointers to all the actual records with that particular srch key value.

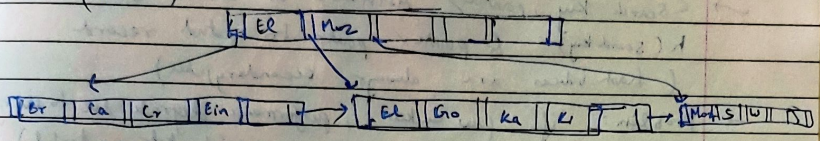
B+ Tree Index Files :
→ to overcome performance degradation of indexed sequential file as file grows and periodic reorganisation of file
⇒ all paths from root to leaf are of same length
→ each node not a leaf has between $\lceil n/2 \rceil$ and n children
→ leaf node has between $\lceil (n-1)/2 \rceil$ and $(n-1)$ values

Note :

$K_1 < K_2 < K_3 \dots < K_{n-1}$

→ leaf node pointer to file record with search key K_i.
P_n points to next leaf node in search key order

eg for $(n=6)$:



→ If K search key values, tree height $\leq \lceil \log_{\lceil n/2 \rceil} K \rceil$
→ node size ≃ disk block
insert, delete, update → $O(\log_{\lceil n/2 \rceil} K)$

B^+ file organisation : same except leaf nodes store records instead of pointers

B^- tree index files :
→ search key values appear once
→ additional pointer in non-leaf nodes included (points to file records)
→ may use less nodes than corresponding B^+ tree, search key value may be found before reaching leaf node. (But B^+ is better)

STATIC HASHING :

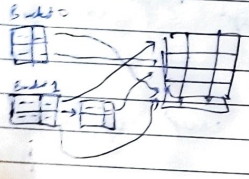
→ bucket = block → contains one or more records
→ hash fn : h (search key) = bucket address then search bucket for record
→ Hash fn should be uniform, random
→ Multiple records may have same search key & different search key may point to same bucket. ⇒ bucket overflow
Sol^n : bucket chaining (linked list)

Hash Indices : (Ordered index part 6 over)

→ hashing can be used not only for file, but also for index creation.

→ <Search key, pointer> distributed in buckets based on h(search key) & pointer : pointer to actual record (hash indices are always secondary index)

→ Problem : if few buckets, performance degrade due to overflow
else too much space wasted (buckets underful)



Dynamic Hashing :

→ Extendable Hashing :

→ use only a prefix of hash fn to index into a table of bucket addresses

→ i bit prefix ⇒ bucket address table size = 2^i $\quad i \leq 32$

→ No. of buckets also change dynamically (splitting of buckets)

DBMS.

M T W T F S S
Page No.:
Date:
YOUVA

→ Data —process→ Information
(raw bytes) (for decision making)

Database → how & where data is stored (storage)
DBMS → database + set of programs
→ access, add, update, delete

Advantages of DBMS over normal file system
→ efficient access (queries)
→ no data inconsistency & redundancy
→ atomicity, concurrency, integrity, durability
(debit + credit) (two debits at an atm)
→ security
→ Data abstraction (different views of the same database)

* schema = design

3 schema Architecture:
→ Data abstraction in 3 levels

i) Physical level → how data is actually stored
(data structures used, compression, encryption, etc)
→ goal: efficient storage & access

ii) logical level / conceptual level → what data is stored (attributes)
(eg: student data → name, id, address, phone no.)
→ constraints (eg: id can't be null, phone no. = 10 digits)
→ relationship b/w data (eg: student table → course table)

iii) View level / External level: same database having different views for different end users
→ eg: Customer database: Logistic dept → Name, Phone, address
Customer service → Name, Phone, product ordered

Data models :-

- way to describe the design of DB at logical level
- describe data, relationships, constraints
- eg: ER model, Relational model, object oriented model, object-relational model

Database languages :-

DDL → create table
DML → select, update, add, delete
SQL → both features present in single language

* How to access database from Application programs?
API : eg C++ → SQL
API → ODBC

* Database administrator → central control of data

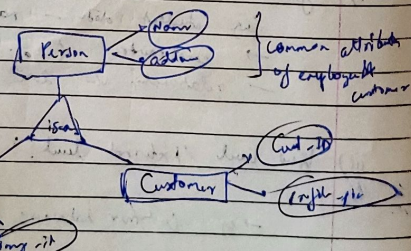
Architecture

T1 → client + server + database in same machine
T2 → client → server + database
T3 → client → application server → database server
used for large scale apps (www)
additional layer (eg: filter malicious queries)

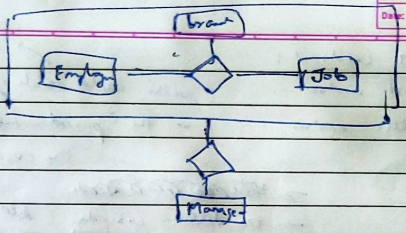
Extended ER features :

i) Specialization :
→ top down
→ is_a relationship

ii) Generalization :
- bottom up. Converse of specialization
- is_a
- Working method is diff but ER diagram is same



ii) Aggregation



Manager manages aggregation of (Emp, Branch, Job)

Steps to make ER diagram :

i) Identify entity & its

ii) " attributes & their types (single valued, multivalued, derived, atomic, composite)
- primary key

iii) " relationship & constraints
↳ mapping (1:N or N:M)
↳ Participation (total / partial)
(strong / weak relationship)

Relational Model : — Implemented through RDBMS (eg SQL, oracle)

(Convert ER digram in form of tables / relations)

* Keys : Primary, Super, candidate, alternate, foreign key
{Candidate} - {primary}

(combination of attributes to uniquely identify a tuple)

(remove redundant attributes from super)

Foreign key : attribute which is a primary key of another relation

eg: Orders relation:

Order_Id	product	date	amount	Cust_id (foreign key)

* Integrity constraints : eg data types (phone_no → integer)
→ primary key constraint (PK != NULL)
→ ~~domain~~ restrict the values (eg age > 18)

M T W T F S S
Page No.:
Date:
YOUVA

* Referential Constraint :
— insert constraint (on the child table / referencing table) → ORDERS
— delete constraint (on the parent table / referenced table) → CUST-MSTR
— on delete cascade (delete value from parent ⇒ delete corresp. entry from child table)
— on delete NULL (put NULL on corres. child table)

* Key Constraints :
→ Not Null (column should not be NULL)
→ Unique constraint (a table may have multiple attributes as unique but will have only one Primary key)
→ default constraint (default value of a column)
→ Check constraint (limit value range (domain))

Converting ER to Relational Model :
→ Composite attribute : add multiple columns (add_city, add_street,)
→ multivalued attribute : add a new relation (cust-id, depndt)
→ Generalization → 3 tables / 2 tables
→ Weak entity → add primary key of strong entity
→ Aggregation → new table with combination of primary key of all participating relations
→ Unary relation : same table → add foreign key (1:N)
or new table (M:N)

eg:

Emp-ID	Role	Manager_emp-ID (Foreign key)
101	SDE	103
102	SDE	103
103	EM	Null

→ Derived attribute → discard (1 manager can manage multiple people)
→ strong entity → relation with attributes as columns

M T W T F S S
Page No.:
Date:
YOUVA

→ SQL :

→ Functional dependency :

AIM : reduce redundancy, insert anomaly, update anomaly, delete anomaly, size of DB
improve access time

1NF → all atomic attributes, no multivalues
2NF → no partial dependency : if AB → C then A ↛ C
(prim, non-prim)
3NF → N- nonprim → nonprim dependency
BCNF → A → B : A must be superkey
(A must determine all the attributes)