① Introduction to ORDB

1 NF to 3 NF

ID	Name	phoneN	ChildID

→

ID	Name

ID	phoneN

ID	ChildID

ORDB → OOC + Relation + DBMS

OOP	ORDB
Class A	Type t
Attributes, Methods	Attribute, Methods
objects:- a_1, a_2, a_3, a_4	objects:- $t_1, t_2, t_3 \ldots$
a_1. Attribute a_1. Method	

* In here, you should store objects in some Storage area. That Storage we are calling Table.
* we use some Language for Store that data. It calls OR-SQL
* When we use OR-SQL
  - we can modify data
  - easyly generate reports.

ex:-

Student_t

Sno
Sname
GPA

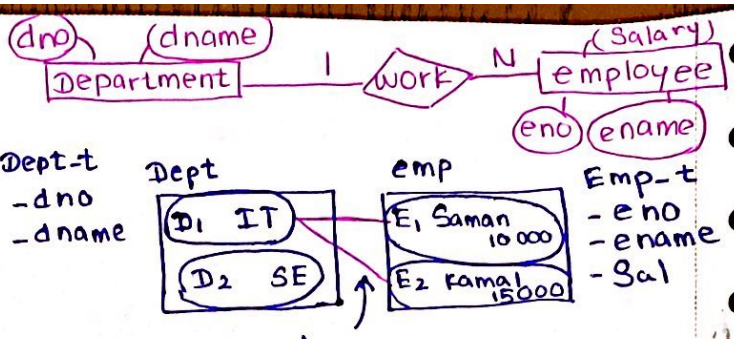
Student

8123	5356
Saman	Kamal
2.5	3.6

Create type:-

```
Create type student_t as object (
      Sno   CHAR(5),
      Sname VARCHAR2(30),
      GPA   FLOAT
)
```

* Object type has 3 Components
  1. name
  2. Attribute
  3. Methods



⊙ CREATE TYPE Dept_t AS OBJECT (
    dno CHAR(5),
    dname VARCHAR2(30)
)

⊙ CREATE TYPE Emp_t AS OBJECT (
    eno CHAR(5),
    ename VARCHAR2(30),
    Sal FLOAT
    workdept REF Dept_t
)

⊙ CREATE TABLE Dept of Dept_t
    (dno PRIMARY KEY)

⊙ CREATE TABLE Emp of Emp_t
    (eno PRIMARY KEY)

⊙ Insert into Dept Values
    (Dept_t ('D1', IT))

⊙ Insert into Dept Values
    (Dept_t ('D2', SE))

⊙ Insert into Emp Values
    (Emp_t ('E1', 'Saman', 10000,
    Select REF(d)
    FROM dept d
    where dno = 'D1'))

⊙ Select eno, e.ename, e.workdept.dname

Co-dependent Type

If there have Many REF & we Can not Identify what type create first. That situation we use this type.

⊙ create empty type.

02. ORDB - Collections.

⊙ VARRY

```
CREATE TYPE price_arr AS
  VARRY(10) OF NUMBER(12,2)

CREATE TABLE pricelist(
  Pno integer,
  Prices price_arr
);

Insert into Pricelist
  Values (1, price_arr(2.50,3.75,4.25));
```

Pricelist

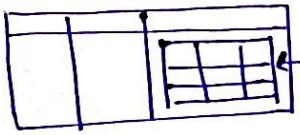
Pno	Prices
1	2.50 | 3.75 | 4.25
2	
3	

⊙ Retrieving from a VARRY

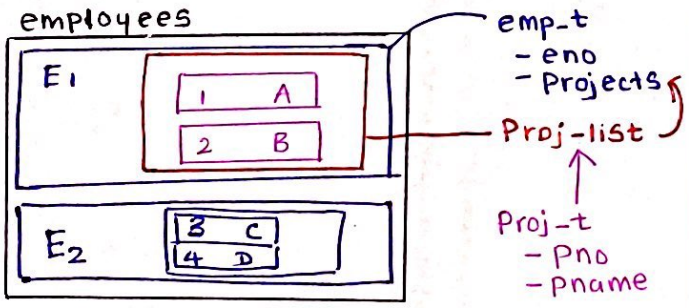
```
Select Pno, S.COLUMN_VALUE Price
FROM Pricelist P, TABLE (P.Prices) S;
```

Pno	Prices
1	2.50
1	3.75
1	4.25

⊙ Nested Tables → one table have another table



ex :-



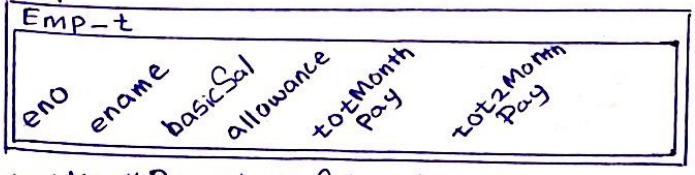
* proj-list → One emp do many projects.

```
*. CREATE TYPE Proj_t AS OBJECT(
     Pno NUMBER,
     Pname VARCHAR2(15));

* CREATE TYPE Proj_list AS TABLE
   OF Proj_t ;

* CREATE TYPE employee_t AS OBJECT(
     eno number,
     Projects Proj_list
   );

* CREATE TABLE employees of employee_t
   (eno PRIMARY KEY)
   NESTED TABLE projects AS
       employees_Proj_table ;

* INSERT INTO employees values
   ( Emp_t (E1, Proj_list (Proj_t
     (1,'A'), Proj_t (2,'B') )));

* . UPDATE employees e
     SET e.projects = Proj_list (Proj_t
                      (1,'B'))
     WHERE e.eno = E1 ;

* . DELETE TABLE (
     SELECT e.projects
     FROM ... )P
```

03. Methods & Inheritance.

*. Member Method

Emp



totMonthPay = basicSal + allowance

```
⊙CREATE TYPE BODY emp-t AS
  MEMBER FUNCTION totMonthPay() RETURN
  FLOAT IS
     BEGIN
       RETURN Self.basicSal + Self.allowance;
     END;
  END;

⊙CREATE TYPE emp-t AS OBJECT (
     eno CHAR(5),
     ename VARCHAR2(30),
     basicSal FLOAT
     allowance FLOAT
     MEMBER FUNCTION totMonthPay() RETURN
     FLOAT
);
⊙CREATE TABLE emp of emp-t
  /
⊙Insert into emp values
  (emp-t ('E123','Saman', 100000 ,20000))
  /
  Insert into emp values
  (emp-t ('E124', 'kamal', 150000, 30000))
  /
⊙SELECT e.ename, e.totMonthPay()
  FROM emp e
  /
```

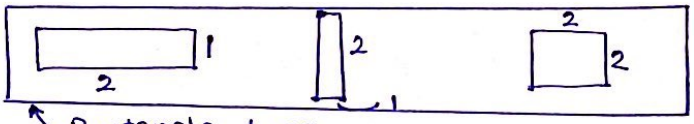
→

EName	E.totMonthPay
Saman	120 000
Kamal	180 000

If you want to add another Member function,

```
⊙ALTER TYPE emp-t
    ADD MEMBER FUNCTION
    tot2MonthPay() RETURN FLOAT
  CASCADE;
⊙CREATE OR REPLACE TYPE BODY emp-t
  AS MEMBER FUNCTION totMonthPay() RETU
     RN FLOAT IS
     BEGIN
       Return self.basicSal + Self.allowance;
     END;
MEMBER FUNCTION tot2MonthPay() RETURN
FLOAT IS
     BEGIN
       Return 2*(self.basicsal+self.allowance);
     END;
END;
```

* map Method - Compare large No of objects



↖ Rectangle_type

```
⊙CREATE TYPE Rectangle-type AS OBJECT
 (   length number,
     width number,
     MAP MEMBER FUNCTION area return
     number
 );
⊙CREATE TYPE BODY Rectangle_type AS
  MAP MEMBER FUNCTION area return
  number IS
    BEGIN
      RETURN length*width ;
    END area;
  END;
⊙CREATE TABLE Rectangles of Rectangle-type;
⊙Insert into Rectangles values (2,1);
                               (1,2);
                               (2,2);
```

```
⊙Select DISTINCT value(r) FROM Rectangles
  r;                    ↳ Duplicate values are removed
```

VALUE(R)	(LEN, WID)
RECTANGLE_Type	(1, 2)
"	(2, 2

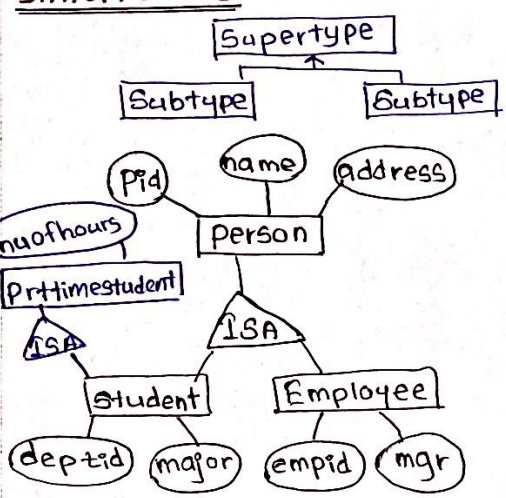
*Order Method - Compare only 2 objects

ex:- Compare Customers by CustomerID.

```
⊙CREATE TYPE Customer_t AS OBJECT
 (  id number,
    name varchar2(20),
    addr varchar2(30),
    ORDER MEMBER FUNCTION match
   (c Customer_t) RETURN Integer);
⊙CREATE TYPE BODY Customer_t AS
  ORDER MEMBER FUNCTION match
 (c Customer_t) RETURN Integer IS
  BEGIN
    IF id < c.id THEN Return -1;
  ELSE IF id > c.id Then Return 1;
    ELSE Return 0;
  END;
END;
```

(current object, Given object id)

Inheritance



⊙CREATE TYPE Person_type AS OBJECT
(Pid number,
name varchar2(30),
address varchar2(100)) NOT FINAL;

⊙CREATE TYPE Student_type UNDER Person_type
(deptid number,
major varchar2(30)) NOT FINAL;
/

⊙CREATE TYPE Employee_type UNDER Person_type
(empid number,
mgr varchar2(30)) NOT FINAL;
/

⊙CREATE TYPE PartTimeStudent_type UNDER Student_type
(numhours number);

⊙CREATE table person_tab of person_type
(Pid Primary key);

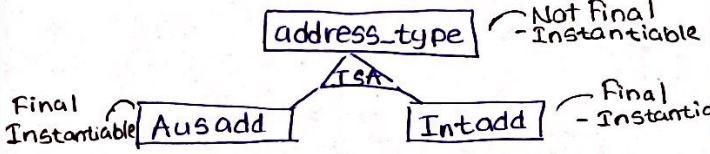
⊙Insert into person_tab values
(Student_type (4, 'Edward learn',
('ABCDAddress', 40, 'CS')
);

Selecting all instances
⊙ SELECT Value(s)
FROM Person_tab S
where Value(s) IS OF (Student_type);
↳ display all details Student_type + Part-time Student

Selecting instances
⊙ SELECT Value(s)
FROM person_tab S
where Value(s) IS OF (ONLY Student_type);
→ display Only Student_type details

Selecting Subtype Attribute
⊙ SELECT Name,
TREAT (Value(p) AS PartTimeStudent_ty
).numhours hours
FROM person_tab P
WHERE Value(p) IS NOT (ONLY PartTime
Student_type);

NOT Instantiable → Specialized subtypes



⊙CREATE TYPE Address_type
AS OBJECT (. . . . .)
NOT INSTANTIABLE NOT FINAL;*

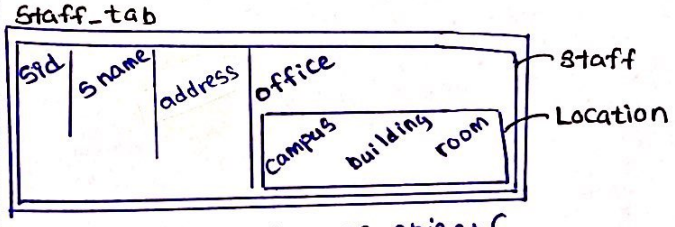
⊙CREATE TYPE AusAddress_type
UNDER Address_type (. . . .);

⊙CREATE TYPE IntAddress_type
UNDER Address_type (. . . .);

N.I. method
⊙CREATE TYPE T AS OBJECT (
x number,
NOT Instantiable Member function
func1()* Return number
) NOT INSTANTIABLE NOT FINAL;

04. Triggers & ER-ORDB Mapping.

Staff_tab



- Create type location as object (
  Campus varchar2(20), building number,
  room number);
- Create type staff as object (
  Sid number, name varchar2(100),
  address varchar2(100), office location);
- Create table staff_tab of staff (
  Sid primary key);

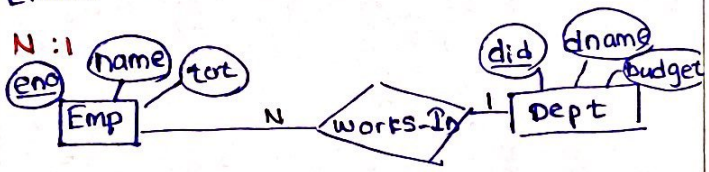
Movement

Sid	Oldoffice	NewOffice
	Location	Location

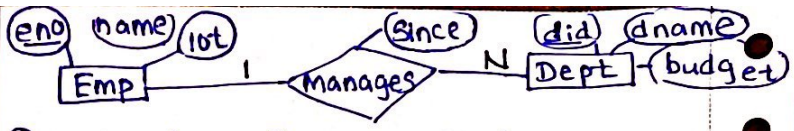
- Create table movement (
  Sid number, old-office location,
  new-office location);

```
CREATE TRIGGER trig1
BEFORE UPDATE OF office ON Staff_tab
FOR EACH ROW
WHEN (new.office.campus = 'Bentley')
BEGIN
   IF : new.office.building = 314 THEN
   INSERT INTO movement values
   (:old.Sid , :old.office , :new.office
   );
   END IF;
END;
```

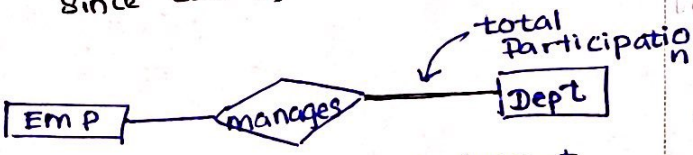
N : 1



⊙Create type Emp_t
(eno char(11),
  name char(20),
  lot Integer,
  workdept ref dept_t
)



⊙Create type Dept_t as object
(did Integer,
  dname char(20),
  budget real,
  mgr ref emp_t
  Since date);



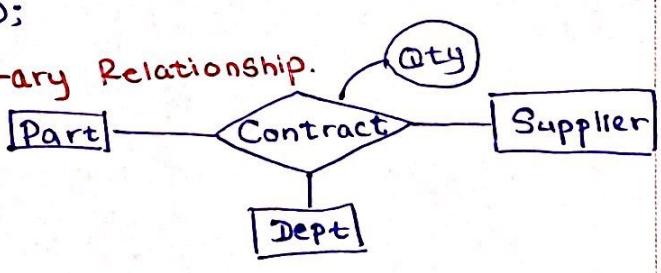
⊙Create table dept of dept_t
(did Primary key,
  mgr NOT NULL References Emp
);
↑ Emp table

M:N

⊙Create type manages_t as object (
  emp Ref emp_t
  dept Ref dept_t
);

Create table manages of manages_t
(emp References Emp
  dept References Dept
);

N-ary Relationship.

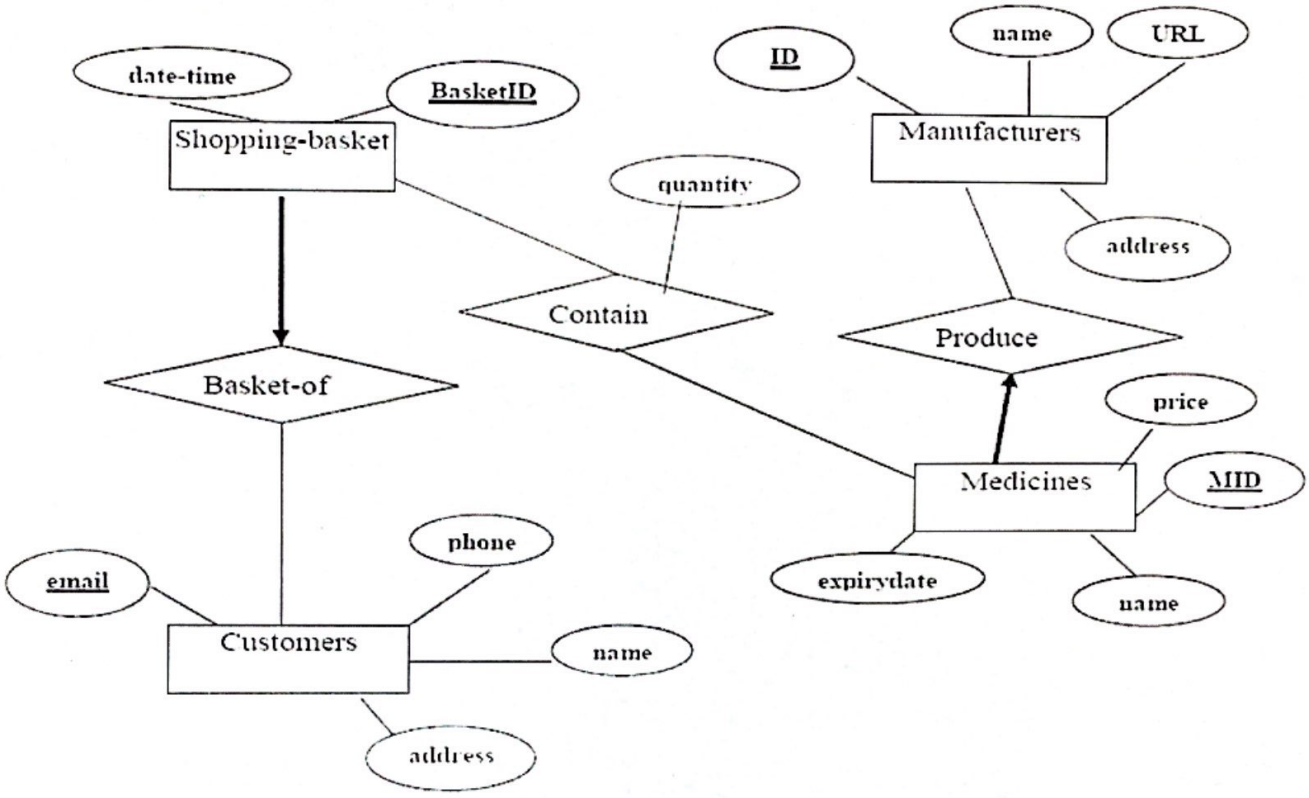


⊙Create type Constract_t as object
(part REF part_t
  Supplier REF Supplier_t
  Dept REF Dept_t
  qty number (6,2)
);

QUESTION 1

The Entity-Relationship model given below shows a part of the database requirements of an on line pharmacy that sells over the counter medications (that do not require a doctor's prescription). Map this ER model to an Oracle object relational schema. Show the steps in your mapping. Indicate all REF types, as well as primary key and referential constraints.

The built-in data types need not be specified. Also, it is not necessary to write CREATE TYPE or CREATE TABLE statements.



Step 1: Map regular entity sets to types:
basket_t (bid, datetime)
customer_t (email, name, address, phone)
medicine_t (mid, name, expirydate, price)
manf_t(id, name, url, address)

Step 2: Map binary relationships with key constraints:
basket_t (bid, datetime, customer ref customer_t)
medicine_t (mid, name, expirydate, price, manufacturer ref manf_t)

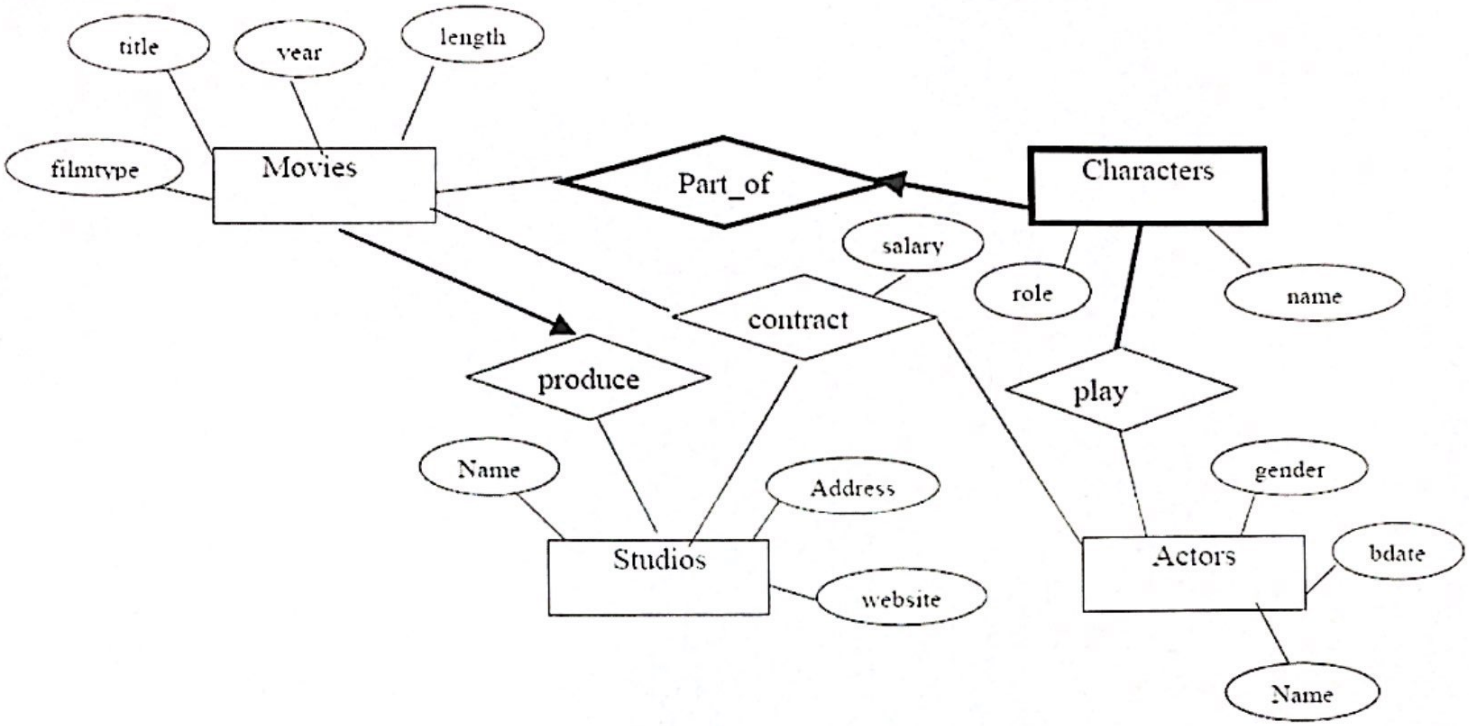
Step 3: Map other binary relationships:
contain_t (basket ref basket_t, medicine ref medicine_t, quantity)

Final set of types:
basket_t (bid, datetime, customer ref customer_t)
customer_t (email, name, address, phone)
medicine_t (mid, name, expirydate, price, manufacturer ref manf_t)
manf_t(id, name, url, address)
contain_t (basket ref basket_t, medicine ref medicine_t, quantity)

Set of tables with keys:
baskets of basket_t (bid primary key, customer not null references customer)
customer of customer_t (email primary key)
manufacturers of manf_t (id primary key)
medicines of medicine_t (mid primary key, manufacturer not null references manufacturers)
contain of contain_t (basket not null references baskets, medicine not null references medicines)

QUESTION 2

Map the following ER model to an Oracle object relational schema. No need to write Oracle SQL statements, but indicate primary and foreign keys and all REF types. Built in data types of attributes need not be specified.



Mapping to Oracle object relational model:

(a) Regular entity sets to types:

Movie_t (title, year, length, filmtype)
Studio_t (name, address, website)
Actor_t (name, gender, bdate)

(b) Weak enity type:

Character_t (movie ref movie_t, name, role)

(c) Binary relationship with key constraint:

Movie_t (title, year, length, filmtype, studio ref studio_t)

(d) Other binary relationship:

Play_t (character ref character_t, actor ref actor_t)

(e) Ternary relationship:

Contract_t (movie ref movie_t, studio ref studio_t, actor ref actor_t, salary)

(f) Final set of types:

Studio_t (name, address, website)
Actor_t (name, gender, bdate)
Movie_t (title, year, length, filmtype, studio ref studio_t)
Character_t (movie ref movie_t, name, role)
Play_t (character ref character_t, actor ref actor_t)
Contract_t (movie ref movie_t, studio ref studio_t, actor ref actor_t, salary)

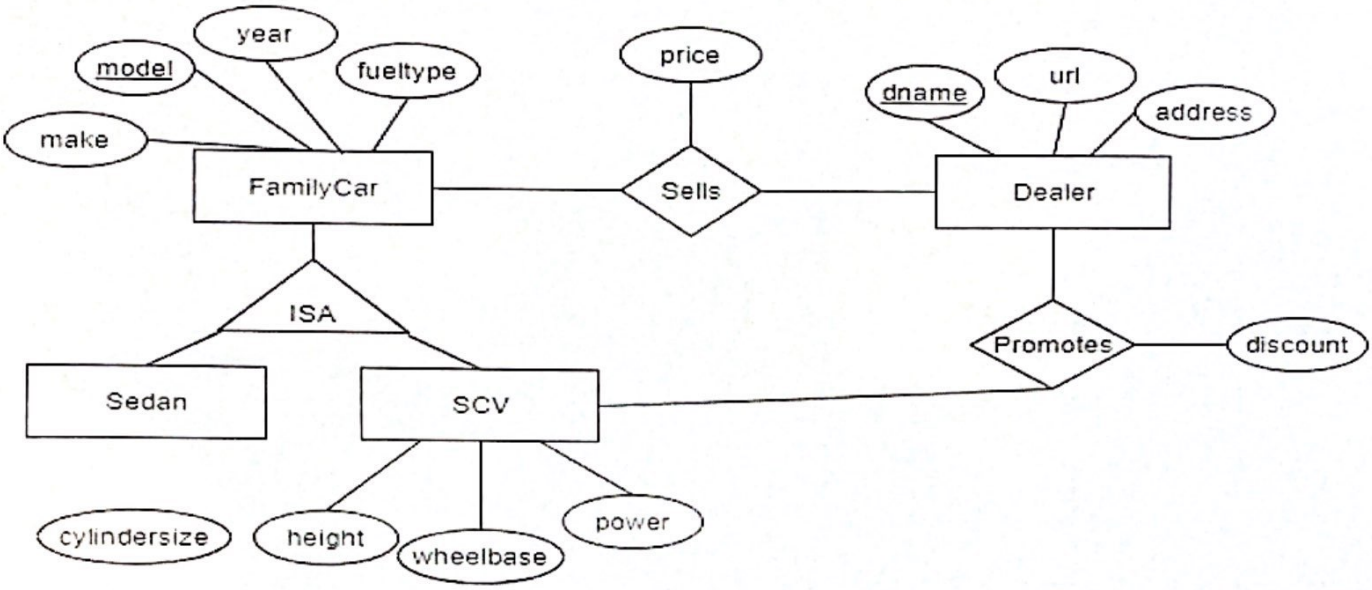
(g) Tables:

Studios of studio_t (name primary key)
Actors of actor_t (name primary key)
Movies of movie_t (title primary key, studio references studios)
Characters of character_t (movie references movies)
Play of play_t (character references characters, actor references actors)
Contracts of contract_t (movie references movies, studio references studios, actor references actors)

QUESTION 3.

The Entity-Relationship model given below represents part of an automobile distribution database. Map this ER model to an Oracle object relational schema. Show the steps in your mapping. Indicate all REF types, primary keys and referential constraints.

The built-in data types need not be specified. Also, it is not necessary to write CREATE TYPE or CREATE TABLE statements.



Step 1: Map entity sets familycar_t (make, model, year, fueltype) not final dealer_t (dname, address, url) **Step 2: Map ISA hierarchy** sedan_t under familycar_t (cylindersize) suv_t under familycar_t (height, wheelbase, power) Step 3: Map m:n binary relationship types sells_t (familycar ref familycar_t, dealer ref dealer_t, price) promote_t (suv ref suv_t, dealer ref dealer_t, discount) **Final set of types:** familycar_t (make, model, year, fueltype) not final dealer_t (dname, address, url) sedan_t under familycar_t (cylindersize) suv_t under familycar_t (height, wheelbase, power) sells_t (familycar ref familycar_t, dealer ref dealer_t, price) promote_t (suv ref familycar_t, dealer ref dealer_t, discount) (Note: ref suv_t is not specified in promote_t since there will be no table of suv_t)	**Tables:** familycars of familycar_t (primary key(make, model)) dealers of dealer_t (dname primary key) sells of sells_t (familycar not null references familycar, dealer not null references dealers) promotions of promote_t (suv not null references familycars, dealer not null references dealers)

Tutorial 5 – XML Databases

Consider the following XML document that contains information on a collection of books.

```
'<?xml version="1.0" encoding="UTF-8"?>
<planes>
   <plane>
      <year> 1977 </year>
      <make> Cessna </make>
      <model> Skyhawk </model>
      <color> Light blue and white </color>
   </plane>
   <plane>
      <year> 1975 </year>
      <make> Piper </make>
      <model> Apache </model>
      <color> White </color>
   </plane>
   <plane>
      <year> 1960 </year>
      <make> Cessna </make>
      <model> Centurian </model>
      <color> Yellow and white </color>
   </plane>
   <plane>
      <year> 1956 </year>
      <make> Piper </make>
      <model> Tripacer </model>
      <color> Blue </color>
   </plane>
</planes>')
```

Note that the above XML document stored in *Planes*(*xText XML*) table created in MS SQL Server and it contains an only single record.

a. Write an XQuery (FLWOR) expression to retrieve the models made before 1970. Produce the following output.

```
<oldPlanes>
  <make> Cessna </make>
  <model> Centurian </model>
  <make> Piper </make>
  <model> Tripacer </model>
</oldPlanes>
```

```
SELECT xText.query (' let $planes := /planes/plane
return <oldPlanes>
{
   for $x in $planes
   where $x/year <= 1970
   return ($x/make, $x/model)
```

```
}
</oldPlanes>
')
FROM planes
```

b. What would be the output of the following XQuery expression?

```
SELECT xText.query (' let $planes := /planes/plane
return <results>
{
   for $x in $planes
   where $x/year >= 1970
   order by ($x/year)[1]
   return ($x/make, $x/model,$x/year )
}
</results>
')
FROM planes
```

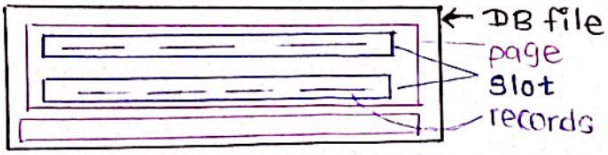
```
<results>
  <make> Piper </make>
  <model> Apache </model>
  <year> 1975 </year>
  <make> Cessna </make>
  <model> Skyhawk </model>
  <year> 1977 </year>
</results>
```

c. Write an XQuery expression to extract data from books XML document and create an HTML table containing the model of all the planes along with their color. Use the following HTML tags in your answer.

```
<table>
  <tr>
    <th>Model</th>
    <th>Color</th>
  </tr>
  … … …
  <tr>
    <td> Tripacer </td>
    <td> Blue </td>
  </tr>
</table>
```

```
select x.query('let $planes := /planes/plane
return <table><tr><th>Model</th><th>Color</th></tr>
{
   for $x in $planes
   return <tr><td>{data($x/model)}</td><td>{data($x/color)}</td></tr>
}
</table>')
from Planes
```

06. File Organization & Indexes



- DB file
  - Search Operation
    - Scan
    - Seek
      - Range – Sorted
      - Equality – Hashed
  - modify Operation
    - Input – Heap
    - Update
    - Delete

file Organization

① Heap – records are stored random order
- Search → (Equality / Range) needs to scan the file
- Insert → at the end of file – fast
- Delete → Search record & Delete

② Sorted / sequential → all records are in some order
- Search (Range) – fast
- Search (Equality)
- Insert – not fast (Sorted order)
- Delete

③ Hashed
- Search (Equality) – fast
- Search (Range)
- Insert
- Delete

Indexes

ex :- Book Index

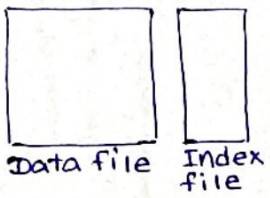
- Alternatives

① Alt 1

age	St. details
age	St. details
age	St. details

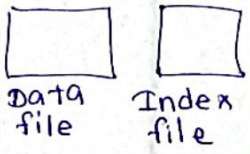
→ Index entry & table records are in same single file

② Alt 2



→ There are 2 different file
→ We should use RecordID to connect these 2 files. recordID = (page#, slot#)
→ To find record, we use recordID & SearchId

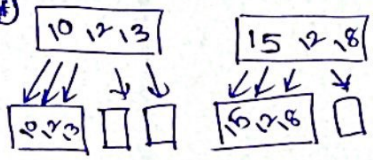
③ Alt 3



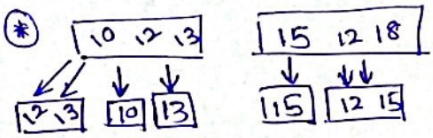
→ In here, we removed redundant Data in Index file.

Clustered & unclustered

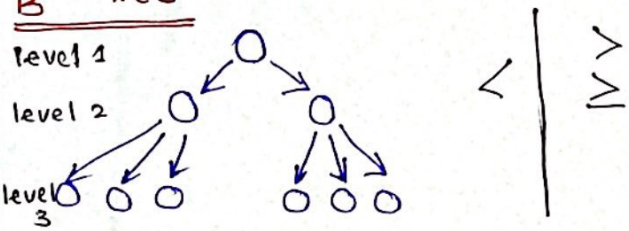
(#)



(*)



B+ Tree



⊙ Search

```
func tree_Search (nodepointer, Search key value k) returns nodepointer
if *nodepointer is a leaf return nodepointer;
else,
  if k < k1 then return tree_Search (p0, k);
else,
  if k ≥ km then return tree_Search (pm, k);
else
  find i such that ki ≤ k ≤ ki+1;
  return tree_Search (pi, k)
  end if
end if
```

⊙Inserting

Find Correct leaf L.
Put data entry onto L.
- If L has enough space, done!
- ELSE, must Split L (into L and a new node L_2)
  - Redistribute entries evenly, Copy up middle key.
  - Insert into entry pointing to L_2 into parent of L.

This can happen recursively
- To Split index node, redistribute entries evenly, but push up middle key. (Contrast with leaf Splits.)

Splits "grow" tree; root split increases height.

Tree growth: gets wider or One level taller at top.

⊙Deleting

Start at root, find leaf L where entry belongs.
Remove the entry
- If L is at least half-full, done!
- If L has only d-1 entries,
  - Try to re-distribute, borrowing from Sibling
  - If re-distribution fails, merge L and Sibling

If merge occurred, must delete entry from parent of L.

Merge could propagate to root, decreasing height.

Hashing
- Satic hashing
- dynamic hashing
  - Extendible hashing
  - Linear hashing.

Static hashing Problems

⊙Create long Overflow chains can develop & degrade performance.
⊙Deletion may waste space

Extendible Hashing

①Global depth of directory

max # of bits needed to tell which bucket an entry belongs to.

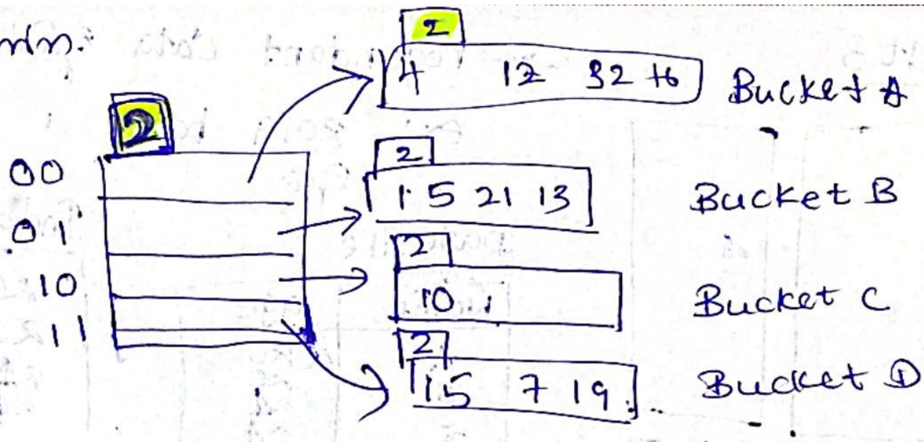
②Local depth of a bucket

of bits used to determine if an entry belongs to this bucket.

Global ≥ Local

① 20 ක් Insert කරන්න.

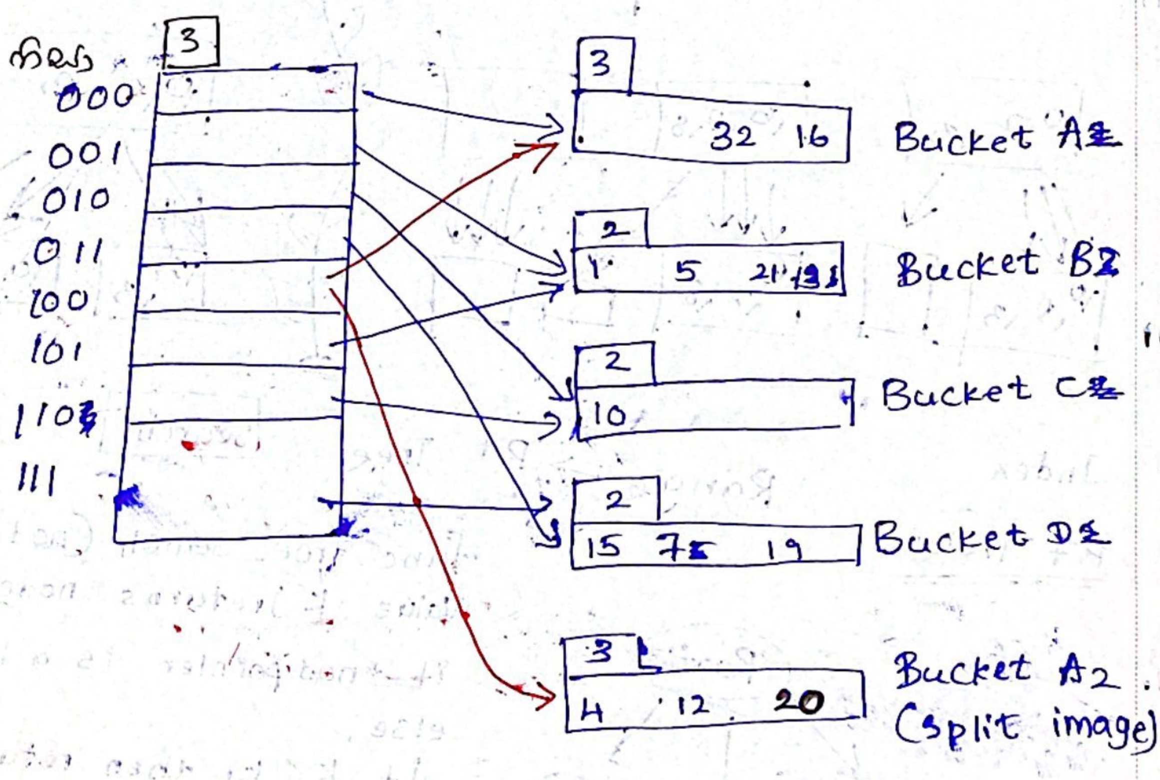
2 | 20
2 | 10 → 0
2 | 5 → 0
2 | 2 → 1
1 → 0



10100

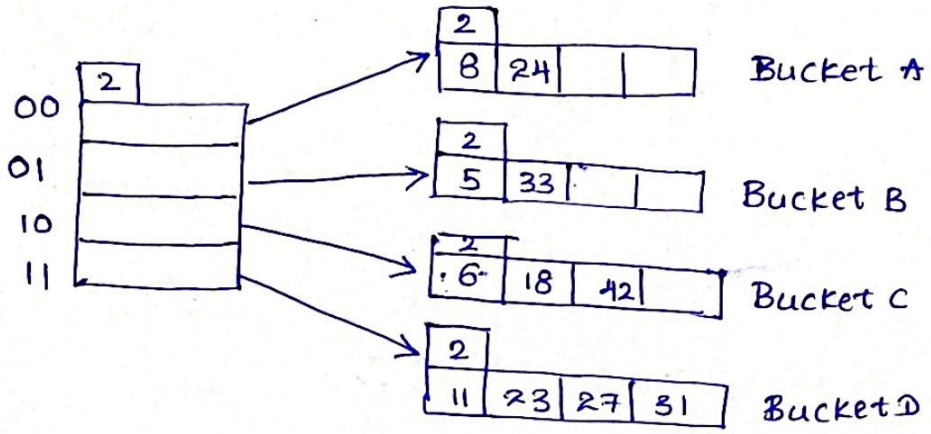
අපි 20 දාන්න ඕනි Bucket A ට But ඒක full එකක්නිසා අපි 00 නොවම 100 තියෙ කියලා හිතනවා එතකොට ~~Global~~ local 1 ද ගොඩක් 3 ක් වෙනවා

local 1 3 ක් නිසා එකම ≥ වෙන්න ඕනිනේ Global 1 එකක්ඩ Global 1 3 ක් වෙනවා



Insert 55

Extedible Hashing

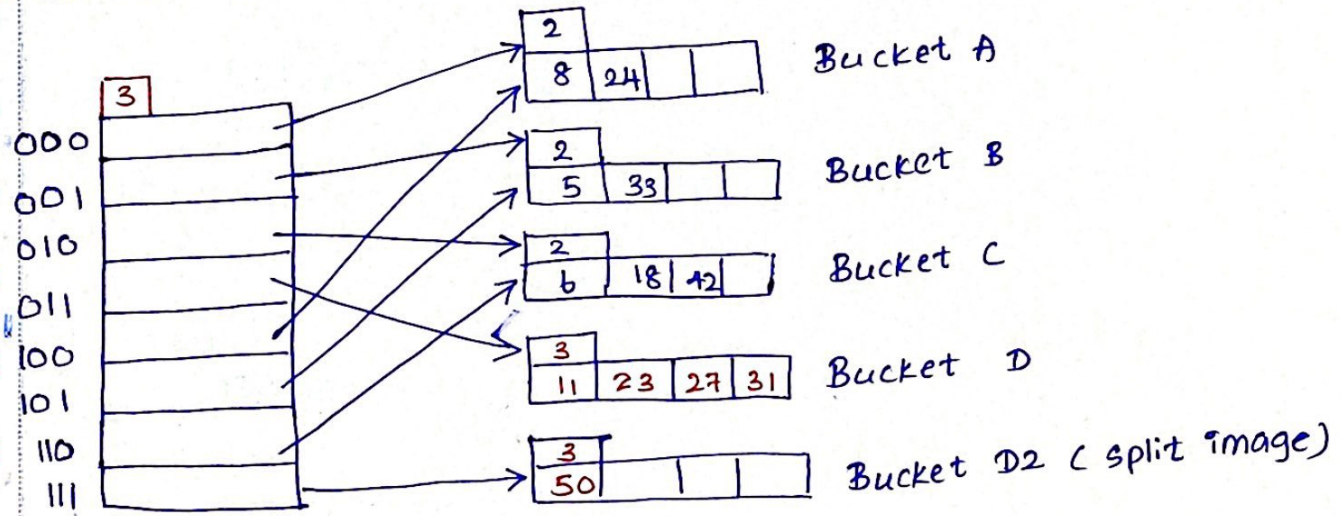


2 | 55
2 | 27 → 1
2 | 13 → 1
2 | 6 → 1
2 | 3 → 0
1 → 1

11011 1

11

11 Bucket D එකට ඇතුල් කිරීම But full. So we create split image. So that we use 111.



07. Query Processing.

Steps in QP

1. Parsing & translation
2. Optimization
3. Evalution

① Equivalence Rules

① Conjective Selection Operation

$$\sigma_{C1 \wedge C2}(E) = \sigma_{C1}(\sigma_{C2}(E))$$

* Rather than evaluating a join Condition together evaluate 1 Condition after the other

② Selection is Commutative – lesser record first

$$\sigma_{C1}(\sigma_{C2}(E)) = \sigma_{C2}(\sigma_{C1}(E))$$

* Both are equal. But it is better to apply the solution first which yield a fewer number of hopes or type of Selection

③ Cascade (π)

* Get Only outermost Projection

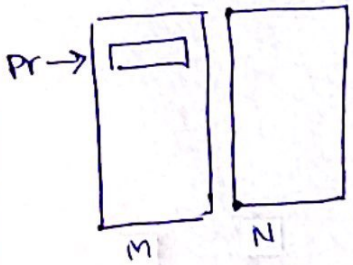
④ Selection Combined with Cartision Product & θ join

* Cross Product Operation is very expensive

② Simple Selection

① Simple nested loop join

Take a record from Outer table R, for each of its record Compare record of inner table S and join



Outer M no of pages

Inner N noof Pages

Cost = $M + M * P_r * N$

② Page-Oriented Nested Loop join

Cost = $M + (M * N)$

③ Block Nested Loops join

Cost = Scan of Outer + #Outer blocks * Scan of inner

$$\#\text{Outer blocks} = \left\lceil \frac{\#\text{ of pages of outer}}{\text{blocksize}} \right\rceil$$

blockzize = available buffers − 2

④ Index Nested Loops join

Cost = $M + (CM * P_r)$ * Cost of finding matching S tuples)

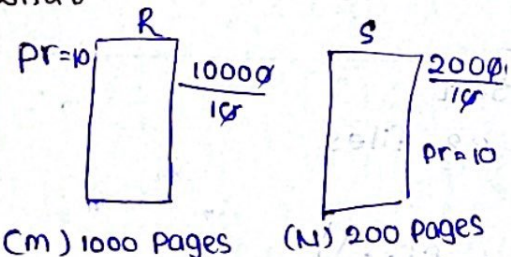
⑤ Sort-Merge join

Cost = $O(M \log M) + O(N \log N) + (M+N)$

Consider the join $R \bowtie_{R.a = S.b} S$; given the following information about the relations to be joined. The cost metric is the number of page I/Os unless otherwise noted, and the cost of writing the result is ignored.

- Relation R contains 10,000 tuples and has 10 tuples per page.
- Relation S contains 2,000 tuples and also has 10 tuples per page.
- Attribute b of relation S is the primary key for S.
- Both relations are stored as simple heap files.
- Neither relation has any indexes built on it.
- 52 buffer pages are available.

a) what is the cost of joining R and S using simple nested loop join?



No indexes

Buffer = 52 pages

R-Outer

S-Inner

* RMS

Cost = M + N x (M x pr)
= 1000 + 200 x (1000 x 10)
= 1000 + 2000 000
= 2001000 I/O

(*) best SMR

Cost = N + M (N x pr)
= 200 + 1000 (200 x 10)
= 200 + 2000 000
= 2000200 I/O

(*) I/O cost is low for SMR ∴ SMR is the best solution. having lesser pages in outer table provides less I/O

b) what is the cost of joining R and S using a page-oriented nested loops join?

best solution

(*) SMR

Cost = N + (M x N)
= 200 + (1000 x 200)
= 200 + 200 000
= 200200 I/O

* RMS

Cost = M + (N x M)
= 1000 + (200 x 1000)
= 1000 + 200 000
= 201000 I/O

c) what is the cost of joining R and S using block nested loops join?

Cost = Max no of Pages of Outer table + (no of blocks in outer table x no of pages in inner)

RMS = $1000 + \left(\left[\frac{1000}{52-2}\right] \times 200\right)$
= 1000 + (20 x 200)
= 1000 + 4000 = 5000 I/O

(*) SMR = $200 + \left(\left[\frac{200}{52-2}\right] \times 1000\right)$
↖ best = 200 + 4000
= 4200 I/O

d) Assuming that there exists a B^+ tree index (with height 3) on b column of relation S, what is the cost of performing an index nested loops join? Explain your answer.

B^+ tree index on b column of relation S ∴ S is inner table

RMS applied assuming ALT 1

Cost = M + innercost × (m × pr)

= 1000 + 3 × (1000 × 10)

= 1000 + 30000

= 31000 I/O

(*) 108 pages, Buffer = 5 external merge sort

Sorting → Pass 0 = $\frac{108}{5}$ = 21.6 = 22 files

marge → pass 1 = $\frac{22}{4}$ = 5.5 = 6 files

Pass 2 = $\frac{6}{4}$ = 1.5 = 2 files

Pass 3 = $\frac{2}{3}$ = 0.66

external merge sort = no of passes × Read or write (1)/ write or read (1)/ write & read (2) × no of pages

= 4 × 2 × 108 = 862

Pass 0 ⇒ $\frac{112}{5}$ = 22.4 = 22

1 → $\frac{23}{4}$ = 5.75 = 6

2 → $\frac{6}{4}$ = 1.5 = 2

3 → $\frac{2}{3}$ = 0.66 = 1

4 × 2 × 112 = 896 ✓

0 → $\frac{72}{5}$ = 14.4 = 14

1 = $\frac{14}{4}$ = 3.5 = 4

2 = $\frac{4}{4}$ = 1

3 × 2 × 72 = 432 ✓

0 → $\frac{120}{9}$ = 13.3 = 13

1 → $\frac{13}{8}$ = 1.62 = 2

2 → $\frac{2}{8}$ = 0.25 = 1

3 × 2 × 120 = 720 ✓

09. Transactions & Concurrency Control

⊙Transactions

A user program may carry out many operations on the data retrieved from the database, but the DBMS is only concerned about what data is read/written from/to the DB.

A Transaction is the DBMS's abstract view of a user program.

A Transaction is a sequence of read & write that belongs to a single unit of work.

Properties of Transaction

① Atomicity

* All action happen or nothing happen
- If completed all actions --> COMMIT (all actions are executed within a DBMS)
- If cannot complete --> ABORT (can undo - none of the actions are executed)

How to maintain atomicity?
- In DB level → using functions & Stored Procedures
- In front end → using connection objects

② Consistency

* Maintain correct data set.
- If user gives correct details the Outp
- it must also be correct.

How to maintain Consistency?
- user must give correct data.

③ Isolation

- DBMS should identify these 2 transactions as to different transaction

How to maintain Isolation?
- developer must set isolation level.

④ Durability

* This property ensures that transaction's survive system Crashed and failures.

uncommitted transcaction is always rolled backed after a system Crash

Transactions & Schedules

* A transaction is seen by a DBMS as a series of ordered actions.

① read
Transaction T reading Object O: $R_T(O)$

② write
Transaction T writing Object O: $W_T(O)$

③ Commit
Commiting transaction T: $Commit_T$

④ Aborting
Aborting transaction T: $Abort_T$

Concurrent Execution of Transactions

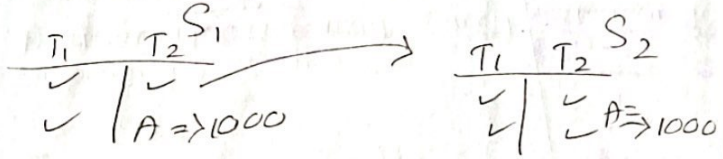
* DBMS interleaves actions of different transactions to improve performance

motivation for Concurrent transactions
- CPU can process one transaction while another is waiting for a page to be read from disk
- Interleaving short transactions with longer transaction allows to complete quicker.

Scheduling Transactions

① Serial Schedule:
- Schedule that does not interleave the actions of different transactions. (No parallel execution)
- effect of serial schedule → DB ends with correct data

② Equivalent Schedules - For any DB state, the effect (on the set of objects in the db) of executing the first schedule is identical to the effect of executing the second schedule.



③ Serializable Schedule - A schedule that is equivalent to some serial execution of the transactions.

Can run transaction parallelly, the effect must be equal to the execution of serial transaction. → Serializability

Anomalies - wR, Rw, ww

(*) →

Lock-Based Concurrency Control

Script Two-phase Locking Protocol - (Script 2PL)

There are 3 rules.

Deadlock

LOG

LSN	
00	begin checkpoint
10	end checkpoint
20	update: T1 write P5
30	update: T2 write P3
40	T2 commit
50	T2 end
60	update: T3 writes P2
70	T1 abort
X	crash, Refresh

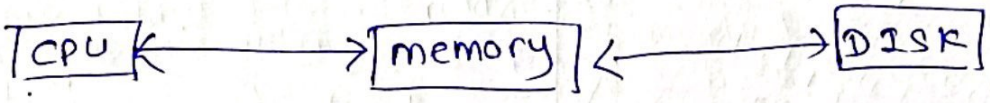
Transaction Table

XID	Status	Last
T1	uncommited	LSN ~~20~~ 70
~~T2~~	~~uncommited~~	~~LSN 30~~
T3	uncommited	LSN 60

Dirty Page Table

Page#	RecLSN
P5	LSN 20
P3	LSN 30

* System does transaction parallelly
* To process we have Only One CPU
* CPU can execute 1 instruction at a time
* Then how does this happen parallelly?



T_1	T_2
R (A)	
W (A)	
	R (B)
	W (B)
R (C)	
W (C)	

Anomalies (*)

① Reading uncommitted Data (WR Conf, "dirty reads").

A - 1000
B - 2000

T_1	T_2
R(A) 1000	
W(A) 900	
	R(A) 900
	W(A) 990
	R(B) 2000
	W(B) 2200
	commit
R(B) 2200	
W(B) 2300	
commit	

WR conflicts

T_1 transfers fund A-B 100

T_2 Calculates interest 10%.

W(A) → 1000 - 100 → 900

W(A) → 900 × 10% → 990

W(B) → 2000 × 10% → 2200

W(B) → 2200 + 100 → 2300

parallel

A → 990 B → 2300

Serial වලින් නම් මෙහෙම

T_1	T_2
1000	
900	
	900
	990
	2100
	2310
2000	
2100	

A → 990
B → 2310

T_2	T_1
1000	
1100	
	1100
	1000
	2200
	2300
2000	
2200	

A → 1000
B → 2300

② Unrepeatable Reads (RW Conflicts)

T_1	R(A)	R(A), W(A), C
T_2	R(A), W(A), C	

T_1	T_2
R(A) 20	
	R(A) 20
	W(A) 19
	Commit
R(A) 19	
W(A) 19	
Commit	

RW Conflicts

A = 20

$T_1 \rightarrow T_2$

A − 1

20 − 1 → 19 → W(A)

③ Overwriting uncommitted Data (WW Conflicts)

T_1	W(A),	W(B), C
T_2	(W(A), W(B), C	

T_1	T_2
W(A) 1000	
	W(A) 2000
	W(B) 2000
	Commit
W(B) 1000	
Commit	

T_1 makes A & B to be 1000

T_1 makes A & B to be 2000

A → 2000

B → 1000