Short Note for Lecture 1

1.Create a type and table

Type is same as class in oop. Also, we create table to store objects.use type to declare blue print

Type >> table >> objects


```
CREATE TYPE student_t AS OBJECT
(
        Sno CHAR(50),
        Sname VARCHAR2(50),
        Addr VARCHAR2(50)
)
/

CREATE TABLE students FROM student_t;
```

SELECT * FROM tab; - View all tables

SELECT type_name FROM user_types; - view all types

DESC student_t;- get type description/details

DESC students: get table details

"/" this forward slash says to execute the previous query

2.Create Objects in the table

Insert into command use to create objects.not save data to table only objects

INSERT INTO students VALUES (student_t('s123','Ishara','Galle'))

/

Show details in the table

Select * from students

Or

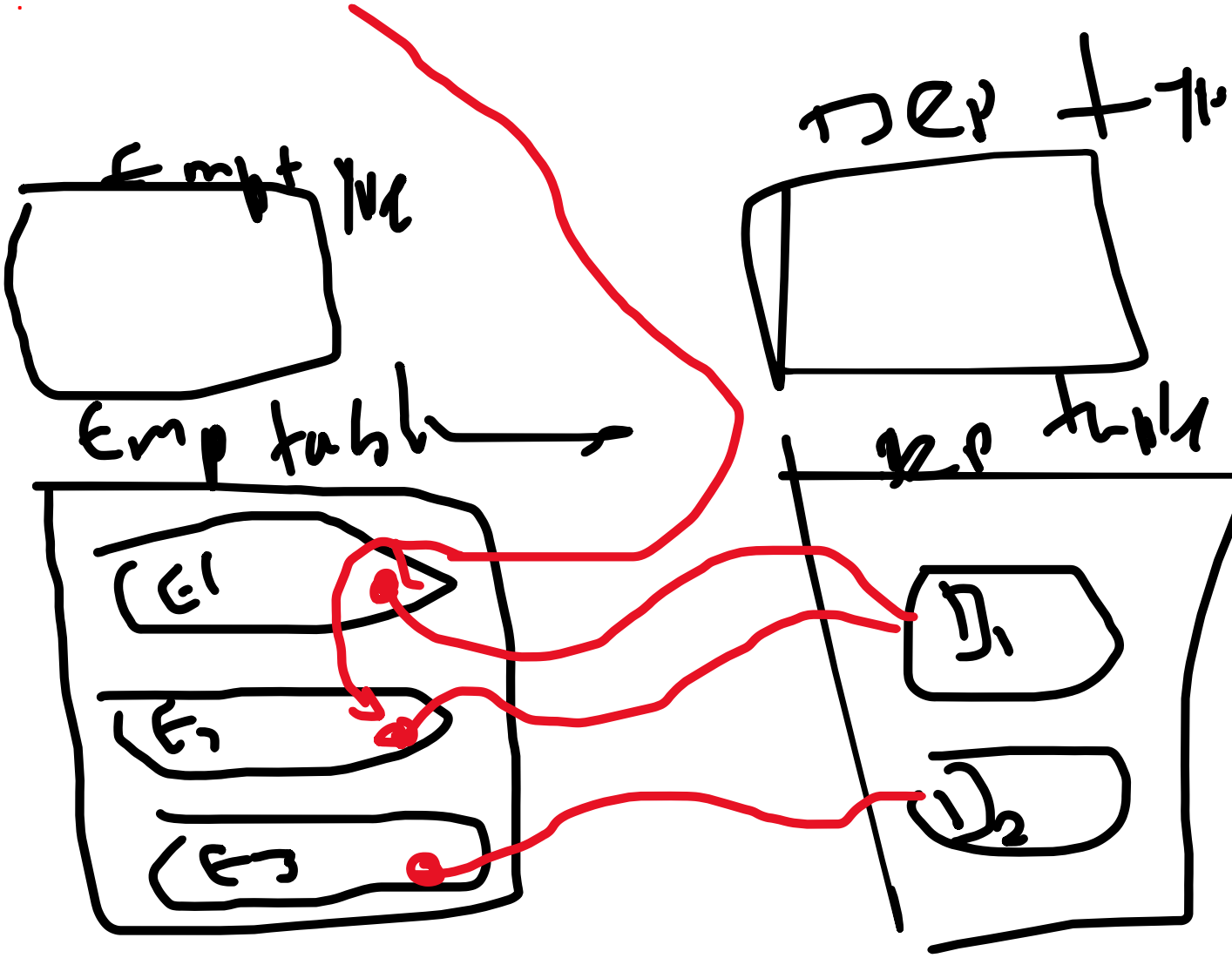
Select value(s) from students s

Get specific columns.

Select s.sno, s.addr FROM students s

Object ID is like a pointer. Use to create relationships between objects

.



Create a object reference to one type object to another

dept

Create type dept_t as object

(

Dno char(5),

Dname varchar2(50)

)

CREATE TABLE dept of dept_t

(

PRIMARY KEY (dno) //if we want we can add a primary key while we are creating tables

)

emp

Create type emp_t as object

(

Eno char(5),

Ename varchar2(50),

Department REF dept_t //store dept_t type object ids

)

CREATE TABLE emp of emp_t

(

PRIMARY KEY (eno) //if we want we can add a primary key while we are creating tables

)

INSERT INTO dept VALUES (dept_t('D1','IT'))

INSERT INTO dept VALUES (dept_t('D2','SE'))

INSERT INTO emp VALUES (emp_t('E100','Ishara','Galle',(

SELECT Ref(d)

FROM dept d

WHERE d.dno = 'D1'

)))

This sub query give the D1's object refference id in department type

Ex -
0000280209A73C3FD13A31498DAF404930FCF10B07A73EE226192F468A872B
B667A061385E0041B9890000

3.Retreive data

SELECT e.eno, e.ename, e.workdept.dname

FROM emp e

WHERE e.ename = 'Ishara'

Or

SELECT e.eno, e.ename

FROM emp e

WHERE e.workdept.dname = 'IT'

Incomplete Types

Short Note for Lecture 2

Collection types 2

1. V array
2. Nested tables

1. V arrays

We use V arrays to store multiple data in same datatype in a single column.

Create Varray

```
CREATE TYPE price_arr AS VARRAY(10) of NUMBER(12,2)
/
```

Use Varray

```
CREATE TABLE pricelist
(
    Pno int,
    Price price_arr
)
/
```

Insert data into Varray

```
Insert into pricelist value(1,price_arr(2.50,3.75,4.25))
```

To retrieve data from this varray as a table we need to convert varray to table.

Convert varray to table

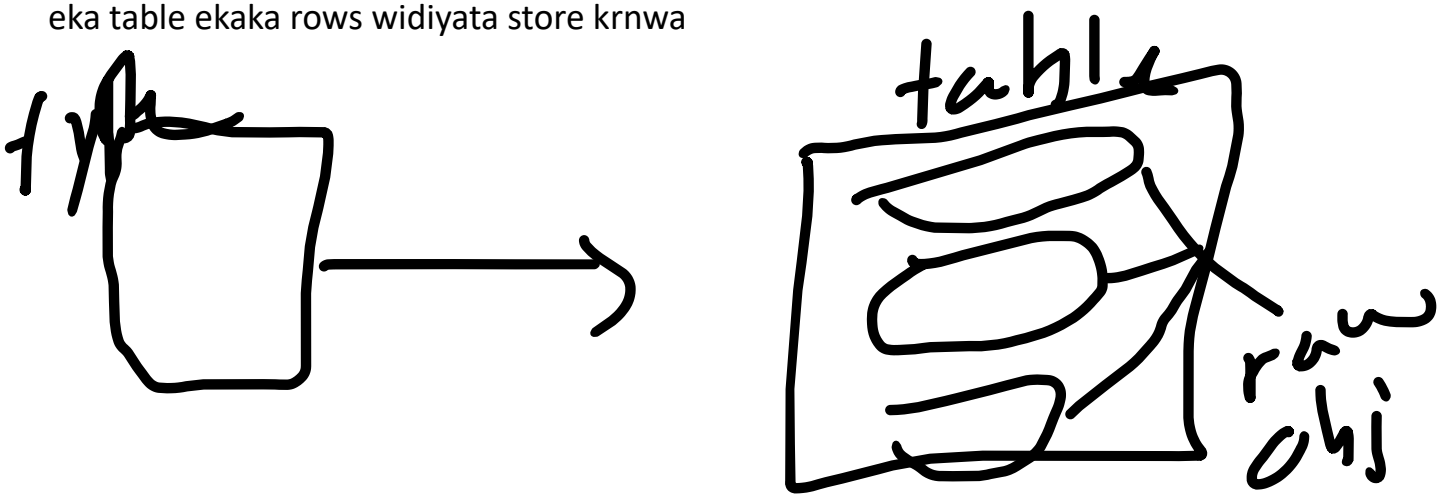
Select pno, s.COLUMN_VALUE price (COLUMN_VALUE kiyane keyword ekak)

From pricelist p, TABLE(p.prices) s (pricelist eke prices kiyanna eka table ekak krala s kiyala ganna. We use table function for this**)**

In object oriented, we can make columns as 2 ways

1. Raw objects
2. Column objects

Raw objects waladi api type eka hadnwa. Ita passe eka use krala object hadala eka table ekaka rows widiyata store krnwa




```
create type emp_t as object(
                empno char(10),
                empname char(10),

        )
        /

        Create table emp of emp_t(
                Primary key(empno)
        )
        /
```

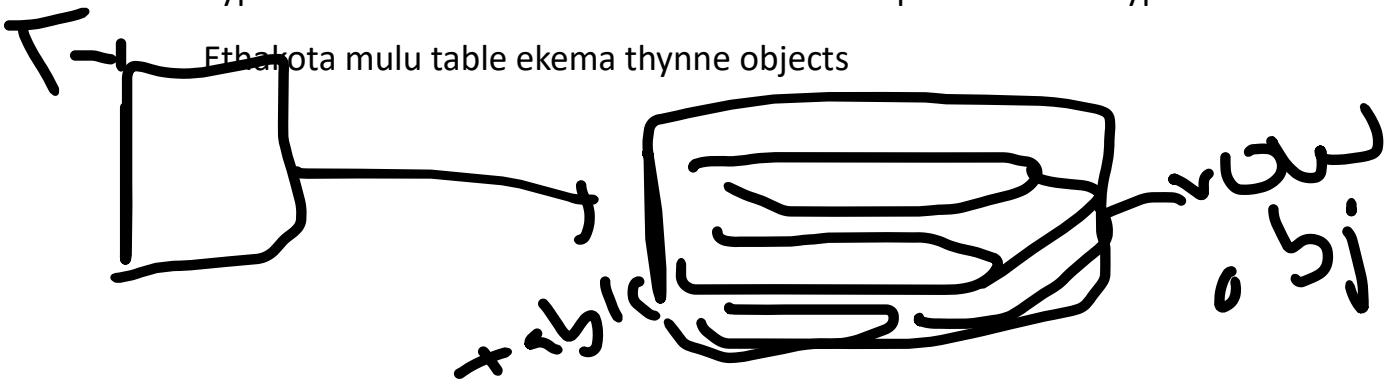
Coumn obj waladi api normal table ekak hdanwa Issara wage. No objects

```
Create table emp(
                Name char(5)
        )
        /
```

Row obj

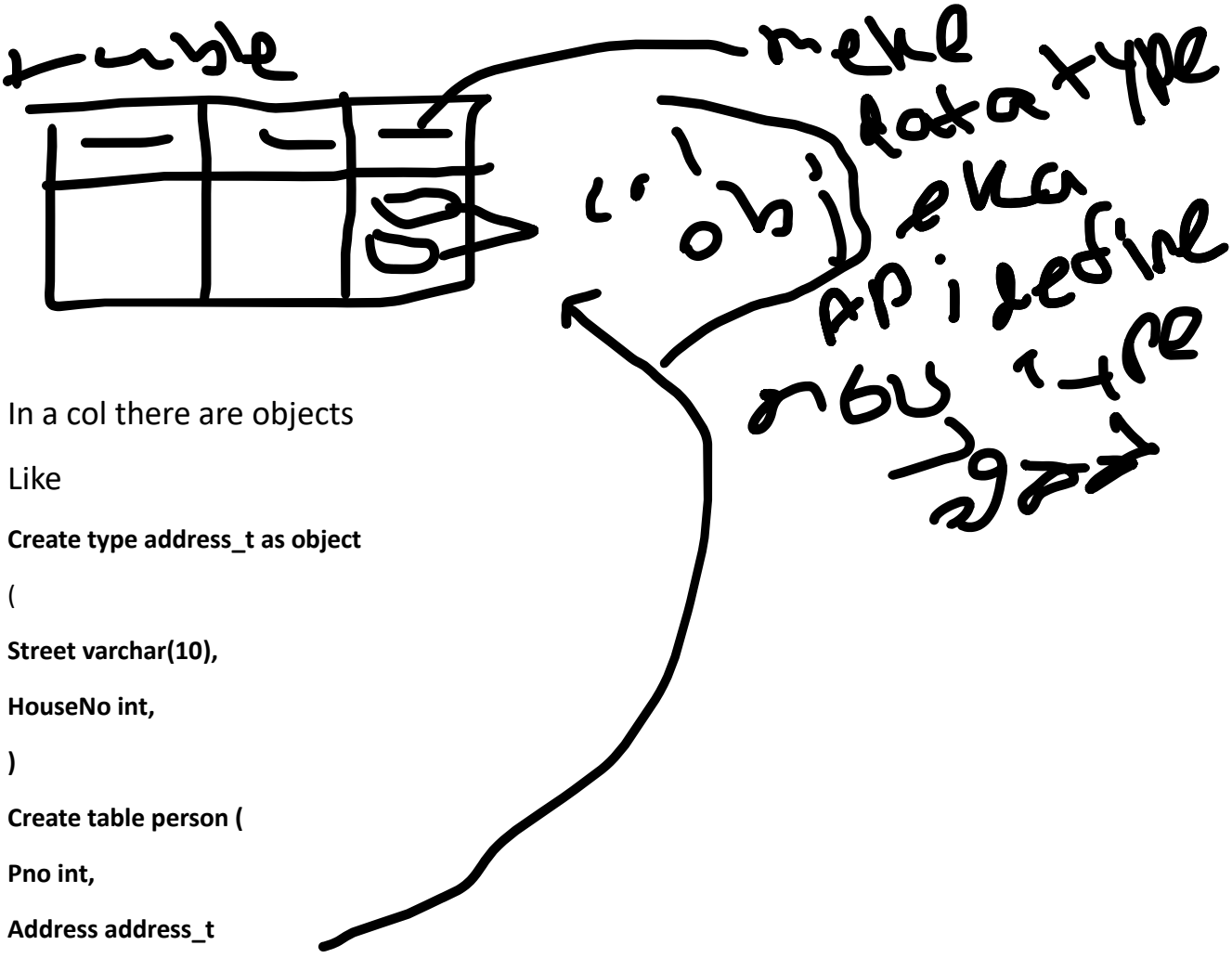
Type ekak hadala eken table ekak haduwoth api data danne type ekata.

Ethakota mulu table ekema thynne objects



Col obj

Table ekak haduwoth api data danne normal widiyta table ekata.
Ethakota one nam column ekak athule apita object hdanna puluwan



In a col there are objects

Like

Create type address_t as object

(

Street varchar(10),

HouseNo int,

)

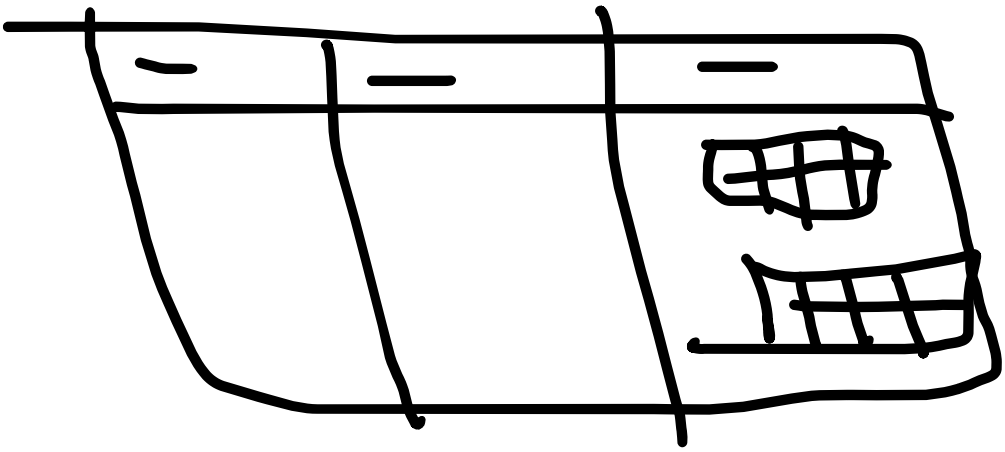
Create table person (

Pno int,

Address address_t

)

2. Nested Tables



Ekkenekta adalawa eyage mokak hari ekak gana data godak thiyaganna nam apita thawa table ekak one. Eka nisa ekata nested table kiynwa.

Employeege project wala wisthara thiyanna one nam api employee athule project kiyala wenama table ekak hdnwa. Object hadala krnna ba. Object ekakta puluwan eka data ekai thiyagnna.E1 ge eka project ekak thiyanna nam raw object ok. But project godak thibboth e row object eke eka col ekak ekak nested table ekak krnna one.

Type ekak hada gnnwa ape project tika store karaganna

```
CREATE TYPE project_typee as OBJECT
(
projectNo NUMBER,
projName varchar(20)
)
/
```

Ita passe eken container ekak hdagnnwa ape projects dala thiyanna

```
CREATE TYPE project_list AS TABLE OF project_type
/
```

Dan thamai hari type eka hadagnne. Empge details danna. Mekn thama object hdanne employeege

CREATE TYPE employee_type as OBJECT

(

Eno NUMBER,

Projects project_list

)

/

Meken thama emp table eka athulata ara api hadagatha table eka nested table ekak widiyata daganne.emp table eka haddima nested table ekath denna one

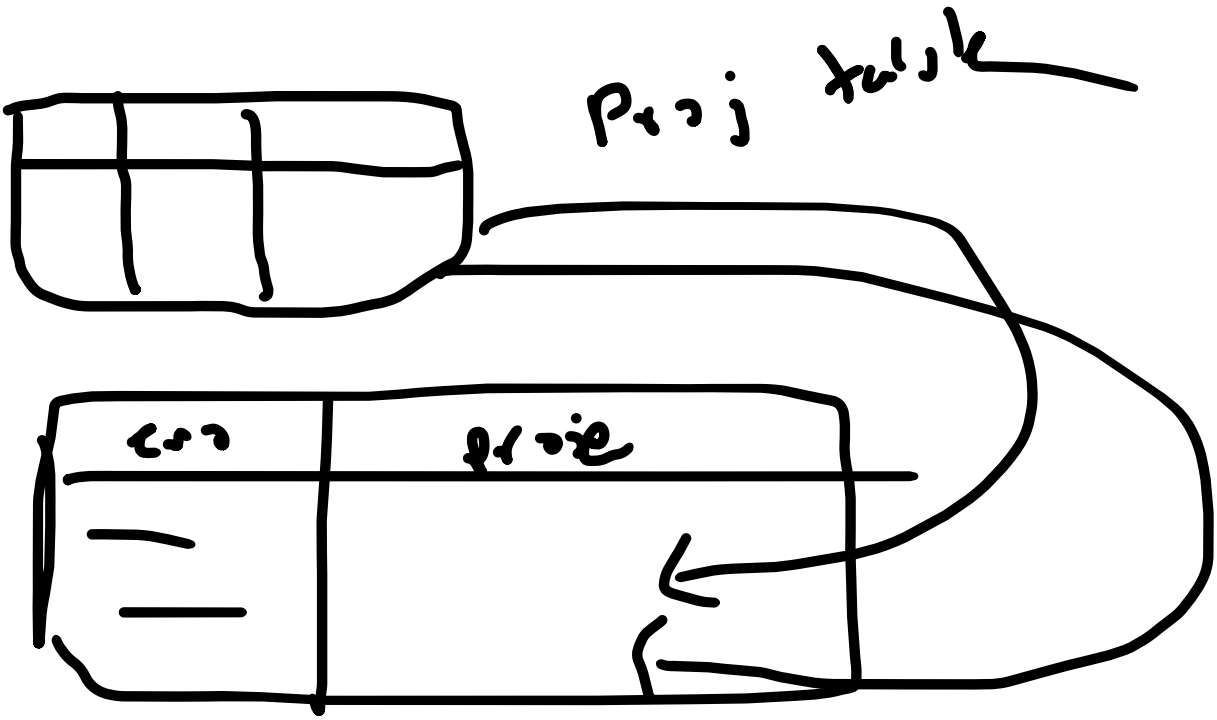
CREATE TABLE employes of employee_type (Eno PRIMARY KEY)

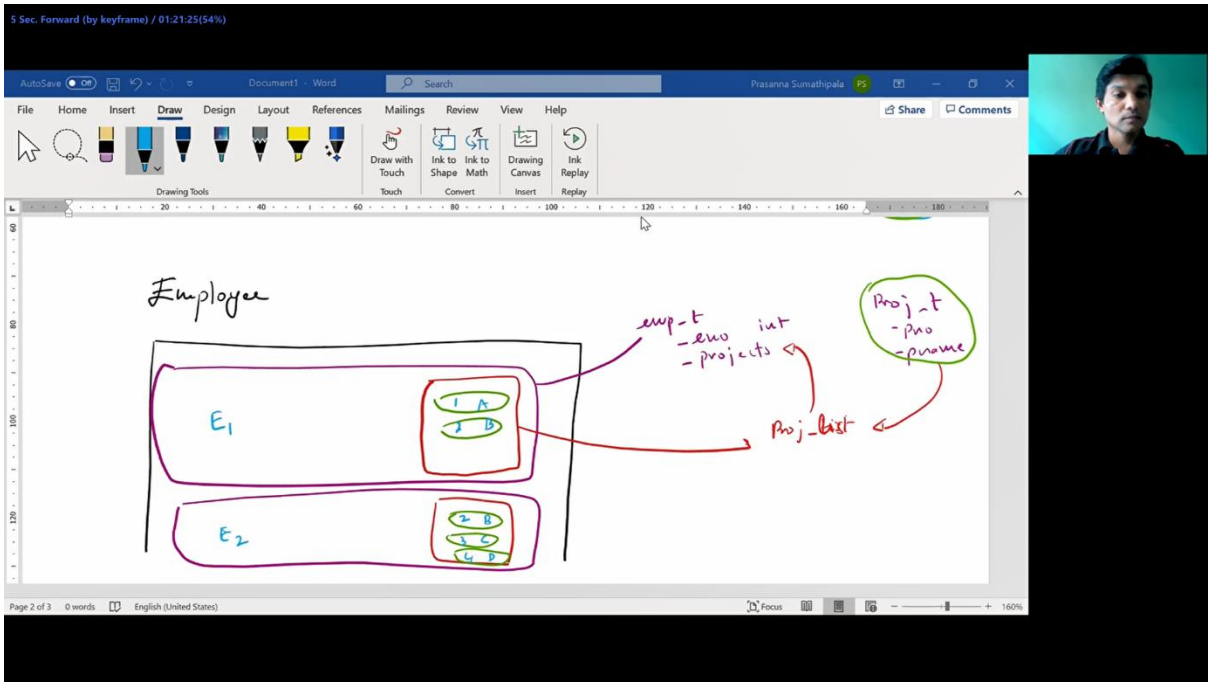
NESTED TABLE projects STORE AS employees_proj_table

/

Mekedi nested table ekanama ganne employees_project_table kiyala

Eka store wene projects kiyan col eka yatathe





INSERT INTO employees VALUES

(employee_type (1000, project_list (project_type(101,'Avionics'), project_type(102,'Cruisecontrol'))))

Employee table ekata data danwa. E daddi eke col ekak thynwa projects kiyala

Ekata ayema ara proj_list kiyala hadagatha table type eka danwa, eka athule data danwa project_t type eken objects widiyata. Project eken ekata wena wenama object thynwa project_t eken. E tika store krpu table type eka thama proj_list

Data retrieve kraddi nested table eke ewa gnnwanm

SELECT *

FROM TABLE(SELECT t.projects FROM employees t WHERE eno = 1000);

Meken enwa eno eka 1000 wena employeege projects tika.

Hama employeegema num ekai projectsnui ganna one nam pahala widiyata

SELECT e.eno, p.*

FROM employee e TABLE(e.projects) p

Insert data into table

```
insert into employees values(
                    employee_type(1000,
                        project_list(
                            project_type(100,'project_1'),
                            project_type(200,'project_2')
)))
/
```

Data insert krnne danta inna kenekta nam issella eyage projects tika aragannwa table eka widiyata ita passe ekata values danwa

```
INSERT INTO TABLE(SELECT e.projects FROM employee e WHERE e.eno =  1000)
VALUES (103, “PROJECT NEPTUNE” )
```

Update nested table data

```
Update table (select projects from employees where eno = 1000) p
Set p.project_name = ‘Avionics’
Where p.project_no = 100
```

Delete nested table data

```
DELETE TABLE (SELECT e.project FROM employees e) p
Where p.project_no = 400
```

To drop a nested table add null

```
UPDATE employees e SET e.projects = NULL WHERE e.eno = 1000;
```

Add back a nested table

```
UPDATE employees e
SET e.projects = proj_list(proj_t(103, 'Project Pluto'))
WHERE e.eno=1000;
```

SQL>@Z:\script – execute script.sql file

SQL>spool z:\prac2.out - By spooling you can direct the output of the SQLPlus editor into a file

SQL> set echo on

SQL> set echo on

time consuming process: you may set termout off and echo on and then run the script to perform the process. So that the script would run as a background process

ORDB: Methods and Inheritance

Member Method

Functions or procedures are declared in an object type definition to implement the behavior of objects.

- Declared in a CREATE TYPE statement and Defined in a CREATE TYPE BODY statement.

Declare member methods

```
CREATE TYPE MenuType AS OBJECT (
        bar REF BarType,
        beer REF BeerType,
        price FLOAT,
        MEMBER FUNCTION priceInYen(rate IN FLOAT)
                RETURN FLOAT)
        /
```

Define member methods

```
CREATE TYPE BODY MenuType AS
MEMBER FUNCTION
priceInYen(rate FLOAT)
RETURN FLOAT IS
        BEGIN
                RETURN rate * SELF.price;
        END;
END;
/
```

```
CREATE TABLE Sells OF MenuType;
Commit ;
```

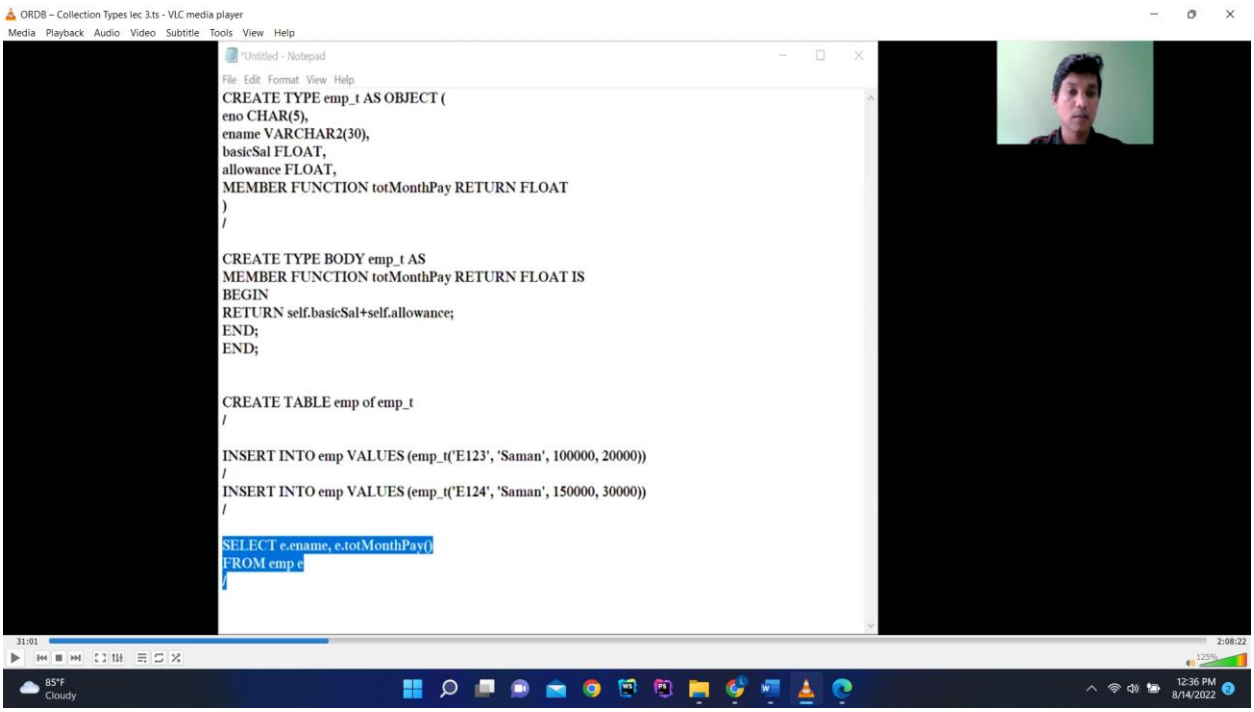
Call member method

```
SELECT s.beer.name, s.priceInYen(106.0)
FROM Sells s
WHERE s.bar.name = 'Joe''s Bar';
```

mehi method ekk create krddi use krn SELF key word ek java vala this key word eka vge. Mehi **priceInYen** method ekt parameter ekk pass krnn oni nattan apita nikmma **priceInYen** kiyl tiynn puluvn. Keep mind parameter pass krnne nattan () varahan danne na method ek declare karan tana. just name ek vitri tiyanne. But method ek call krddi api open close bracket dann oni

SHOW ERROR key word eken apita error ek mkkd kiyla balagan puluvn.
ED key word eken kalin liypu code ekk aran eka edit krnn puluvn.

Example code :



Declare a new member method to an existing object

```
ALTER TYPE MenuType
        ADD MEMBER FUNCTION
        priceInUSD(rate FLOAT)
        RETURN FLOAT
        CASCADE;
```

(Mehi cascade keyword ek anivaryai. Bcz existing ekkt add krn nisa)

Define member methods to existing one

```
CREATE OR REPLACE TYPE BODY
        MenuType AS
        MEMBER FUNCTION
        priceInYen(rate FLOAT)
        RETURN FLOAT IS
        BEGIN
                RETURN rate * SELF.price;
        END priceInYen;

        MEMBER FUNCTION
        priceInUSD(rate FLOAT)
        RETURN FLOAT IS
        BEGIN
                RETURN rate * SELF.price;
        END priceInUSD;

END;
```

(Me vidiyata tavat function ekak define krddi kalin tibb function ekat api aaai liynn oni. nattan error ekk env)

Object Comparisons

Saralavam qwot object compression kiynne object compare krl eva same da, mkkd greater than eka, mkkd less than ek kiyl hoyn ekata. Mekat ORDB vala Map method ek ha Order method ek use krnva

- The values of scalar data types such as CHAR or REAL have a predefined order.
- But instances of an object type have no predefined order.
- To compare two items of a user-defined type, define an order relationship using a map or an order method

1. Map Object

Apita map key word ek use kirimen,

1. Comparisons krnn puluvn, such as obj_1 > obj_2 and
2. DISTINCT, GROUP BY, ORDER BY keyword use krnnt puluvn. (Map keyword eka natuva apita me keyword use krnn ba. Bcz object valata compression krnn reason ekk hadagann ba.)

Map method ekkt parameter tiyenna be

```
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;
```

(Mekat kalin ek vgemai. Venasa tama function ek declare krn tanai define krn tanai function ekt kalin **MAP** keyword ek use krn eka)

incase above code create a compilation error while creating body

```
CREATE TYPE BODY Rectangle_type AS MAP MEMBER
  2  FUNCTION area RETURN NUMBER IS
  3  BEGIN
  4  RETURN length * width;
  5   END area;
  6  END;
  7  /
```

Example: - (Ihata create krpu table ekt data insert krl api example ek krmu)

INSERT INTO rectangles VALUES (1,2);
INSERT INTO rectangles VALUES (2,1);
INSERT INTO rectangles VALUES (2,2);

1. **SELECT DISTINCT VALUE(r) FROM rectangles r;**

Output -> RECTANGLE_TYP (1, 2)
RECTANGLE_TYP (2, 2)

(Attatama meke comparison ek siddavenne area valat. Api samanyayen parameter pass krnne natuva method ekk haduvama ek automa call venva. So, Meka baluvama tereneva api data 3k insert krt print vela tiyenne 2i. bcz mehi first dekema area ek 2i. So Distinct keyword ek nisa duplicated code ain vela tiynva. Itin me vge Distinct, group by, order by keyword use krnn oni nm map function create krnn venva.)

2. **SELECT VALUE(r)**
   FROM rectangles r
   ORDER BY VALUE(r) desc
   /

Value(r) eken venne object ek e vidiytm print karan eka.

2. Order methods

Called automatically whenever two objects need to be compared

Saralavama qwot **ORDER** method eke ha **MAP** method eke tiyena vens nam MAP method ek okkoma object eke compare krl blnva but order method ek object 2k vitri compare krnne. E qwe assume apita oredre by eken object tika descending order ekt hadann oni. Ehidi internally apita object okkoma ekinekata compare krl tama eke order ek hadann oni. So api eyat use krnne **MAP** ek. But assume apita oni object dekak compare krl e deken vishalam ek ganna. Ehidi internally okkoma object apita call krnn avashya na. just e object deka vitrk compare karama ati. E vge velavat api use krnne **ORDER** method ek

Example: -

```
CREATE TYPE Customer_typ AS OBJECT (
id NUMBER,
name VARCHAR2(20),
addr VARCHAR2(30),
ORDER MEMBER FUNCTION match (c Customer_typ) RETURN INTEGER
); /


CREATE TYPE BODY Customer_typ AS
        ORDER MEMBER FUNCTION match (c Customer_typ)
        RETURN INTEGER IS
        BEGIN
                IF id < c.id
                        THEN RETURN -1;
                ELSIF id > c.id
                        THEN RETURN 1;
                ELSE
                        RETURN 0;
                END IF;
        END;
END; /
```

Methods on nested tables

Meke krnne saralavama methods nested table ekak atule implement krn vidiya balana eka.

Example: -

```
CREATE TYPE proj_t AS OBJECT (
        projno number,
        Projname varchar(15)
);
/

CREATE TYPE proj_list AS TABLE OF proj_t;
```

```
CREATE TYPE emp_t AS OBJECT (
        eno number,
        projects proj_list,
        MEMBER FUNCTION projcnt RETURN INTEGER
);
/
```

Meke tiyna aINTO key word ek "=" ekt samana venva. E qwe p.projno count eka pcount valat assign venva

```
CREATE OR REPLACE TYPE BODY emp_t AS MEMBER
        FUNCTION projcnt RETURN INTEGER IS
        pcount INTEGER;
        BEGIN
                SELECT count(p.projno) INTO pcount
                FROM TABLE(self.projects) p;
                RETURN pcount;
        END;
END;
 /
```

```
CREATE TABLE emptab OF emp_t
        (Eno PRIMARY KEY)
        NESTED TABLE projects STORE AS emp_proj_tab;
/




SELECT e.eno, e.projcnt() projcount
FROM emptab e;
/
```

INHERITANCE

ORDB vala object type 4i.

Final or Not Final

instantiable or not instantiable (pahala kata krnva meka gana**)**

By default, ek **Final** ha **instantiable** ve. Apita oni nm eya Not **Final** lesa ho **Not instantiable** define krnn puluvn. To permit subtypes (sub type valat avasara dimata), Inheritance vala all parent objects, not final viya yutui. Sub type final ve.

EX: - **CREATE TYPE Person_type AS OBJECT (pid NUMBER, name VARCHAR2(30), address VARCHAR2(100)) NOT FINAL.**

Apita final or not final lesa switch venn oni nm **ALTER TYPE** keyword ek use krnn puluvn.

Ex: - ALTER TYPE Person_type FINAL;

Creating object

CREATE TYPE Person_type AS OBJECT (pid NUMBER, name VARCHAR2(30), address VARCHAR2(100)) NOT FINAL/

Creating Sub Type object

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)); /

(Mehi api person type ek inherit krl child type dekak hadan tiynva. But mehi student_type not final krl employee_type final krl. Eyata hetuva student_type ek part_time_student type eken aai inherit krn nisa. E qwe student_type kiynnr part_time_student eke parent nisa)

CREATE TYPE PartTimeStudent_type UNDER Student_type (numhours NUMBER);

Creating Table

Table hadaddi apita akara dekkt krnn puluvn. Root level ekata ha leaf level ekt table hadann puluvn. Root level ekt table ekk hadan hati api blmu,

Create table person_tab of person_type (pid primary key).
Create table employee_tab of person_type (pid primary key).

Inserting Data

Insert into person_tab values (student_type(4, 'Edward Learn', '65 Marina Blvd, Ocean Surf, WA, 6725', 40, 'CS'))/

Insert into employee_tab values (Employee_type (4, 'Edward Learn', '65 Marina Blvd, 6725', 40,'CS'))/

Retrieve Data

1. **SELECT VALUE(p) FROM person_tab p;**

   Person_type(21937, 'Fred', '4 Ambrose Street')
   Student_type(27362, `Peter', … , 21, 'Oragami')
   PartTimeStudent_type(2134, 'Jack',…, 13, 'Physics', 5)

   Person_type(21362, 'Mary', …)
   Student_type(18437, `Susan', … , 13, 'Maths')
   PartTimeStudent_type(4318, 'Jill',…, 21, 'Pottery', 2)

   Person_type(39374, 'George', …)

(Using the **VALUE ()** function to select all instances of a supertype)

2. **SELECT VALUE(s) FROM person_tab s WHERE VALUE(s) IS OF (Student_type);**

   Student_typ(27362, `Peter', … , 21, 'Oragami')
   PartTimeStudent_type(2134,'Jack',…,13,'Physics', 5)

   Student_typ(18437, `Susan', … , 13, 'Maths')
   PartTimeStudent_type(4318,'Jill',…,21,'Pottery', 2)

(**IS OF** key word eken krnne boundary ekk dana ek. Meke student type ek ha ehi subtype vala vitrk data retrieve krnna kyai. (From student type and its subtypes))

3. **SELECT VALUE(s) FROM person_tab s WHERE VALUE(s) IS OF (ONLY student_type);**

Student_typ(27362, `Peter', … , 21, 'Oragami')
Student_typ(18437, `Susan', … , 13, 'Maths')

(**ONLY** key word ek specific table eken vitrk data retrieve karai, (From student type but not from subtypes))

4. **SELECT Name, TREAT(VALUE(p) AS PartTimeStudent_type).numhours hours FROM person_tab p WHERE VALUE(p) IS OF (ONLY PartTimeStudent_type);**

NAME Hour

Jack 5
Jill 2

(TREAT () function to make the system treat each person as a part-time student to access the subtype attribute numhours:,)
(apita part-time studentsl 'sub type ekak' vitrk consider krl data retrieve krnn oni nm mema Treat () function ek use kri, saralavam qwot sub type attribute access krnn oni nm api treat function ek use krnva ema nattan treat function ek avashya na)

apita kisima widiyak na sub type ekaka attribute access krnna meka nathuwa. mokda
select numHours from person_tab r where value(r) is of (only partTime_student)
me widiyata access krnna puluwan super type eke ewa withrai. sub type wala one nam treat eka one

1. NOT INSTANTIABLE Types

This is similar to the abstract class in OO.

CREATE TYPE Address_typ AS OBJECT(...) NOT INSTANTIABLE NOT FINAL; *
CREATE TYPE AusAddress_typ UNDER Address_typ(...);
CREATE TYPE IntlAddress_typ UNDER Address_typ(...);

You cannot create instances of the Address_typ only (similar to “abstract classes” in OO). Saralavam qwot Instantiable kiynne we can create an object. Not Instantiable kiynne We can’t create an object. Mehi object kiynne create type object ek nevei. Ek hriyata class ekk vge. Api e class eken hadagann object tama table eke save krnne. Itin ema save krnna object create krnn ba kiyn ek tama mehi kiynne. Asamanyayen Not Instatntaible object ekk Not final viya yutui. Otherwise ek inherit krnn ba.

not instantiable eken apita child la hadla e childlagen object hdnna puluwan. kelinma ara not instatiable eken object hadnna ba

2. NOT INSTANTIABLE Methods

Abstract class ekk abstract method tiynva vge not instantiable type ekk, not instantiable methods tiynva. Mema methods vala body ekk na. subtype eken tama eva override krnne. So not instantiable method ekk not final viya ytui. Otherwise, eva inherit krnn ba. Types vala vge methods vala by default type final venne na. apita final krnn oni nm explicitly ek mention krnn oni. Saralavam qwot method override krnne not instantiable method haraha

Method overriding

EX: -
```
CREATE TYPE MyType AS OBJECT (
   MEMBER PROCEDURE Print,
   FINAL MEMBER FUNCTION foo (x NUMBER))
NOT FINAL; /

CREATE TYPE MySubType UNDER MyType (
   OVERRIDING MEMBER PROCEDURE Print
 ); /
```

(Me example ekt anuva print ek, not instantiable method ekki. So ek override krl tiynva mysubtype type ekedi)

Method overloading

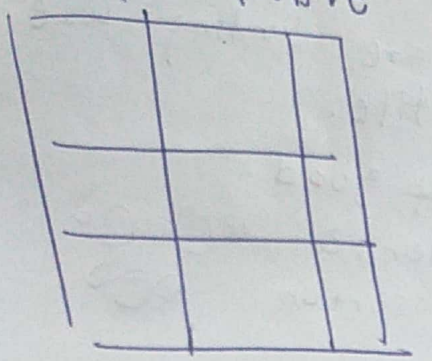
Ex:-
```
CREATE TYPE MyType AS OBJECT ( ...,
   MEMBER FUNCTION fun (x NUMBER)
) NOT FINAL; /

CREATE TYPE MySubType UNDER MyType (
   MEMBER FUNCTION fun (x DATE)
 ); /
```

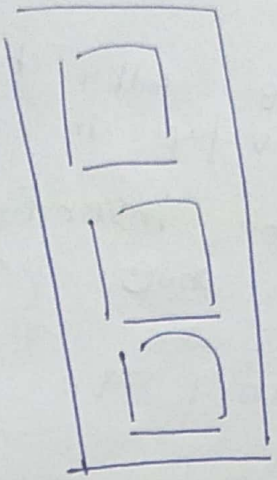
Same function name, different signature

atabase table => DB file.

db Table



DB File



What operations do we have to do with this DB file (read all the table)

-> Search
-> Modify - Insert
update
Delete

① Scanning the table

② seek -> range the table, Eqn.

① Scan -> Assume your table has Columns Sno, Sname, age. for Scanning you want to

```
Select *
from student
```

} for that you have to read all the records of student table.

• Read all - Scan the file Top to bottom

② If you want

```
select *
from student
where sno = 1
```

} you don't need everything. You want to seek some record.

Seek -> range

-> Equality

To perform these operations fast how we should organize our data within this file.

① Heap file organization

There is no specific order of files => Heap

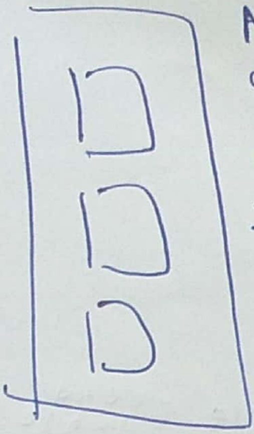
Insert ✓

Search ✗

modify ✗ delete ✗

If records in random order in heap files
What we can perform fast => . Insert

② Sequential



Assume all records are sorting use age value in this file.

✗ for insertion → not good.
Insert කරල පස්සෙ order එක පිළිවෙළට හදන්න ඕන. ඒ නිසා insertion වලට ගැලපෙන් නෑ.

✗ update → update කරන Value එක හොයාගන්න ලේසි. ඒත් Value එක update උනාම පස්සෙ ඒ record එක ආයෙ sorted order එකට හදන්න ඕන.

✗ delete → delete කරල පස්සෙ පිළිවෙළට හදන්න ඕන.

Searching is fast for updating & deleting but after that you want to move records here & there.

✓ Search
→ Scan ✓
→ Seak
— range – sorted files are good for range selection.
When you want to find age between 20 – 40
— equality

Assume you want to search age 20 record
equality වලට හොයන්න හොඳ hash.

③ Hashed

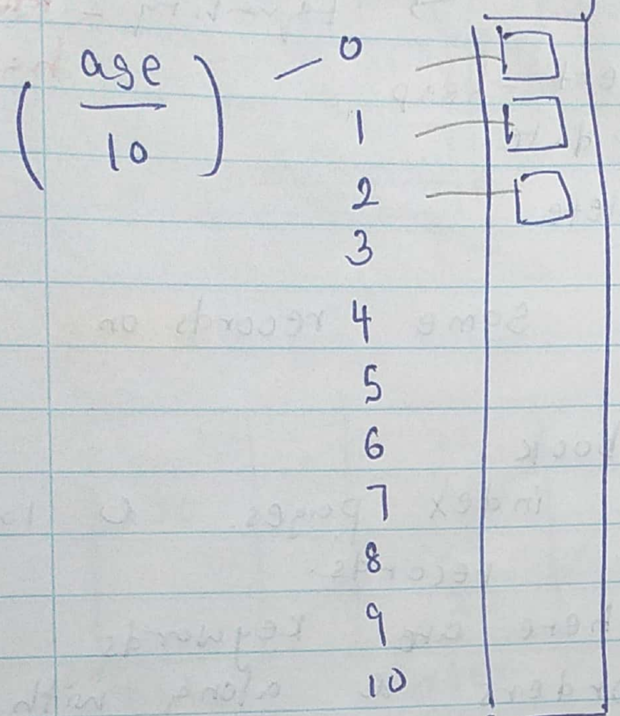
Here it is using some (hash) function

(fh) ← this is organizing records within the file.

Assume my hash function is $\left(\frac{age}{10}\right)$
remainder value

1
2
3
4
5
6
7
8
9

So this hash function tells if your remainder 0, you stay here. (store)



Age = 20 නම් hash function 1 ඇතුළ $\left(\frac{20}{10}\right)$ remainder ඇතය 0 යි.

ඒ ඇතුළ ශූන්‍ය 0ට අදාළ ස්ථානයේ store කරනවා.

මේ particular location 1 ⟹ bucket

- File is a collection of buckets.

h (r) -> bucket in which record r belongs.
hash (r) function එකේ නම, r කියන record 1
ඇතුළෙ එන bucket (location) 1 කියන්නෙ.

Any age record will be directed to any bucket out of those 10 buckets.
So If we have organized all data within this file, equality selection run faster.

Assume we have age = 35 record, apply to hash function, remainder 5 values in one bucket. So we have to read only 1 particular bucket from those 10 buckets.
(Not reading entire file)

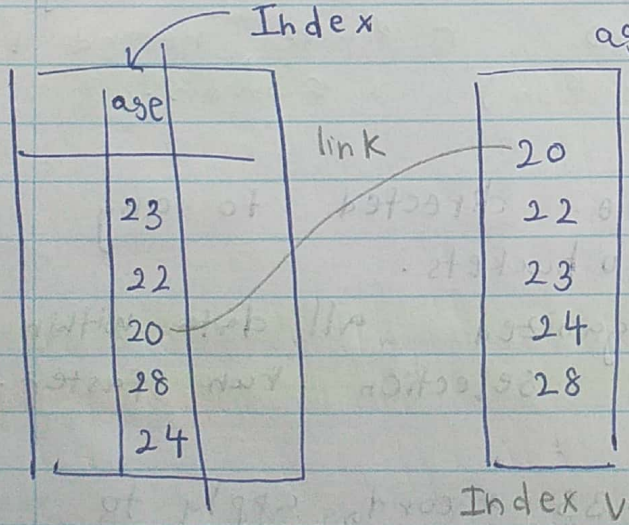
No: Date: __/__/____

Search → Scan → range - sequential
→ seek → equality - hashed

Modify → Insert - heap
update
delete

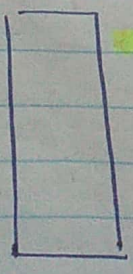
- Assume you have to find some records on a book.
① you can read whole book
② you can go to the index pages & locate to the relavant topic records.
- (In index pages there are keywords with alphabatical orders & along with that some page numbers. so you can straight locate to page number)

- Indexes in the database do the same -



③ different alternatives to available to link those index & table record

Alt 1 → Data record with key value (K)



you have only single file. No db file & index file (no 2 files). Both in same file.

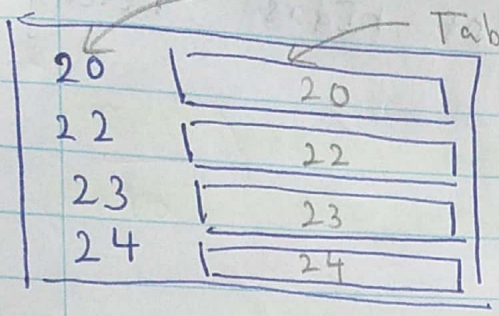
So then this single file you have all age values

Atlas

Date: __/__/____

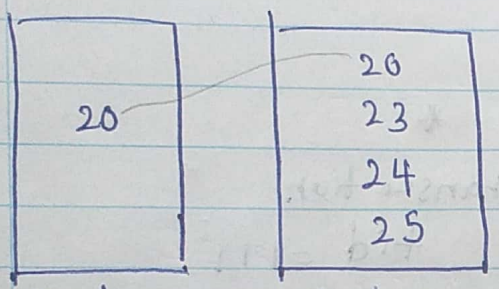
Alt 1

Index entry



Next to age Values record is there.

Alt 2



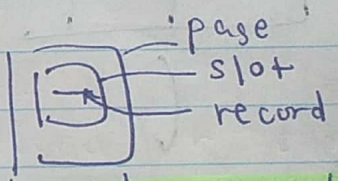
Data file

Index file

There are two files.
- data
- Index

data is in 2 different files. how to connect those index entries & table records.

In alt 2, they maintain Search key along with rId.

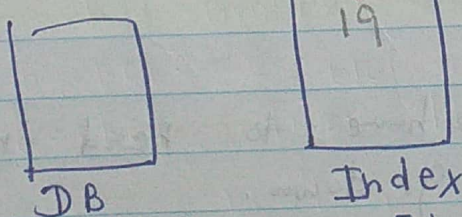


record Id = page # + slot #

Index file ல், Index = 20 அதை தரும் rId = 10. அப்போ, rId ல் இருந்து நமக்கு index ல் page number ம் slot number ம் தெரிந்தால் நாம் நினைத்த record ஐ எளிதாக பெறலாம்.

- If we know rId, we know What is the page & what is the slot.

Alt 3



problem is to handle duplicate data.

Asume this is a age of sliit student so there are plenty of records of age=19 in db record & index file.

(*) Good to use redundant data.

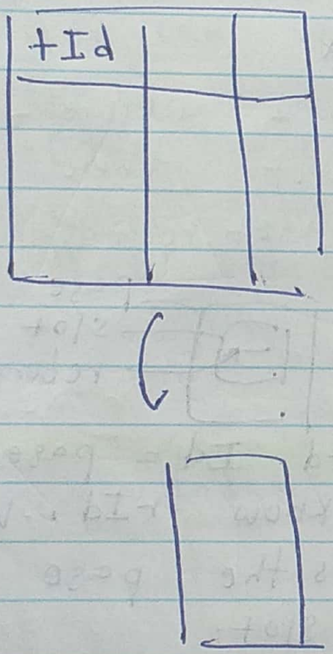
To handle that we (maintain) store 19 once in Index file & maintain list of rId's belong to that age = 19.

Reduce index file size

speed up

(*) Bank System

Transaction Table



Index file

Select *
from transaction
where tid = 111

To speed up this query, we add indexes

- for a one minute this system can have million of transactions.

We are creating indexes for speed up searching But here get in information from index file that also a problem now due to huge number of records.

To find out one entry, you have to read millions of entries. Performance go down.

Index 1 ක [illegible] record 1 [illegible]
record Id 1 [illegible] record, [illegible]

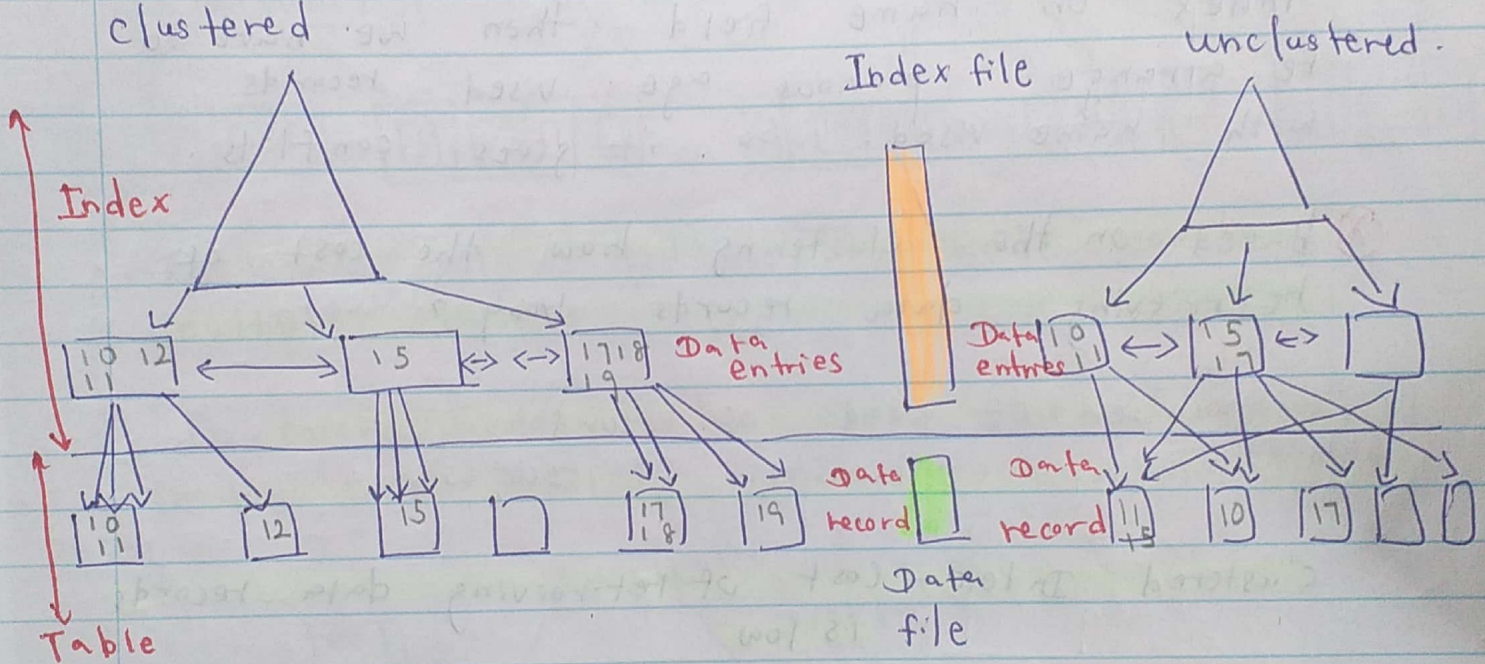
No: Date: / /

without reading millions of entries in index file how to find out particular record.

To speed up, we're developing different data structures.

⇓

Access methods → tree based indexses
hash based indexes.
(how to arrange data within this index file)



data entries => Data entries in index file are
When we create index you copy values from the table column into index file & organize records in the index file.
So all index entries in data entries area.

Data records - full record in the table

(*) If index is clustered, Index entry order & table record order is same sorted order

No :_____ Date: __/__/____

Unclustered - Index entry order & table entry order is not same.

(*) Why we can't have 2 clustered index in table =>

Assume you create index on age column & data records are also sorted as ~~it~~ indexes

If you say you want to add another clustered index on name field, then we have to re arrange previous age vised records with name vised. So it gives conflits.

(*) Based on the clustering how the cost of retrrieving data records vary?

↓

number of reads of records
number of writes of records

Clustered Index - Cost of retreiving data records is low.

Dense ekedi wenne hama data record ekatama index data entry ekak thynwa. eka nisa meka clus or unclus one ekak wenna puluwa.

sparse ekedi wenne hama ekatama data entry ekak na. but canada, cuba kiyana dekatama ekak thynwa c kiyala. ethkota apita cuba one nam c eken gihin blnwa hambenwada kiyala. ehema hambenne nathan c eke ethna indan pahalata search ekak ynwa. eka nisa sparse eke data clustered wenna one aniwa

No:

Date: __/__/____

How to find out index entries

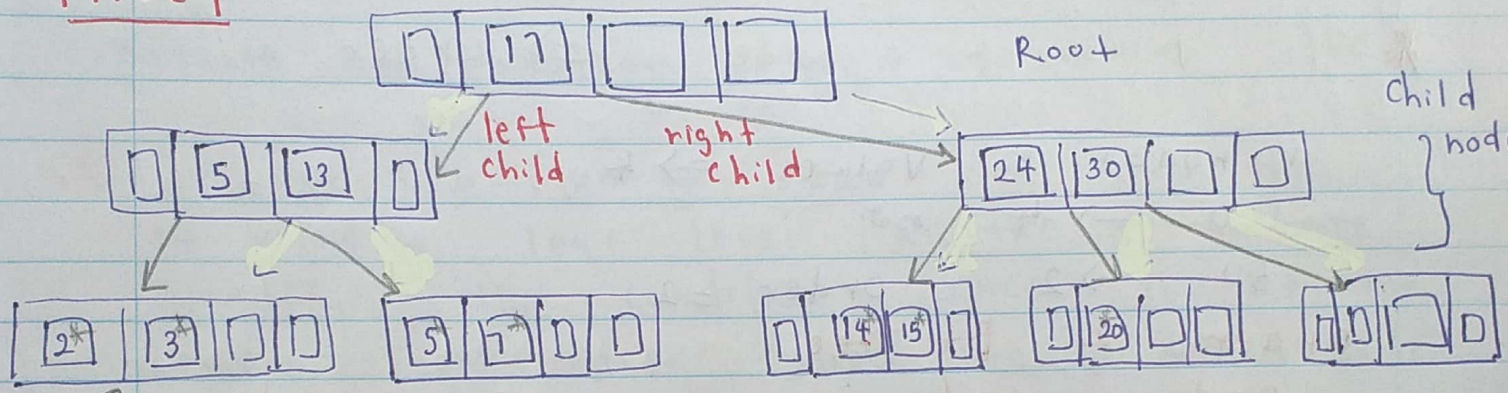
Tree Based Index → B+ Tree

Hash Based Index → E Hash

Tree Based Index → good for range selection

Hash index → Good for Equality selection

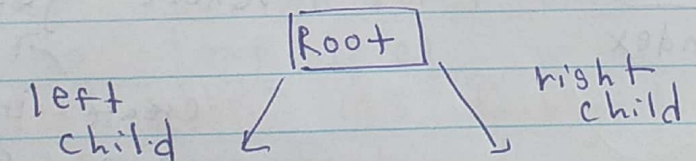
Tree



- Root & all nodes, Each node can have multiple Values

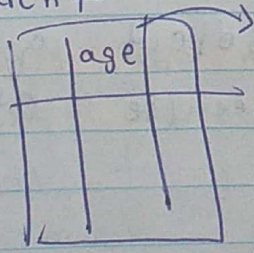
In every node it has left child & right child

In the leaf level all Values are in sorted orden



- less than
- greater than or equal

(*) student



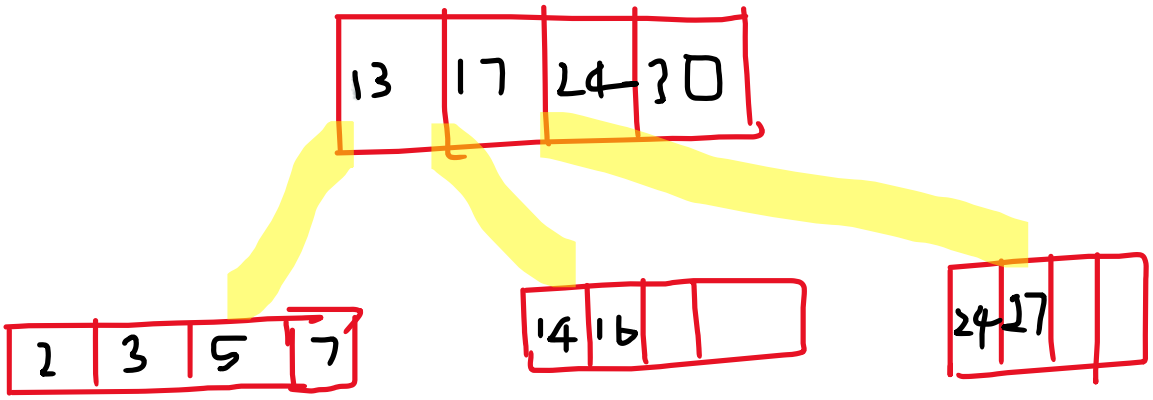
Index is a B+ Tree index. If its a tree structure those age Values are copying to index file.

B+ Tree leaf level have all values took from table column to develop index.

only leaf level has Values came from table column

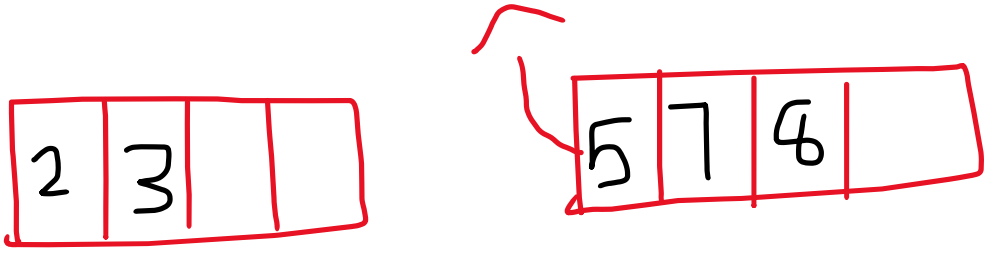
From leaf level we link Actos table records.

B+ tree insert

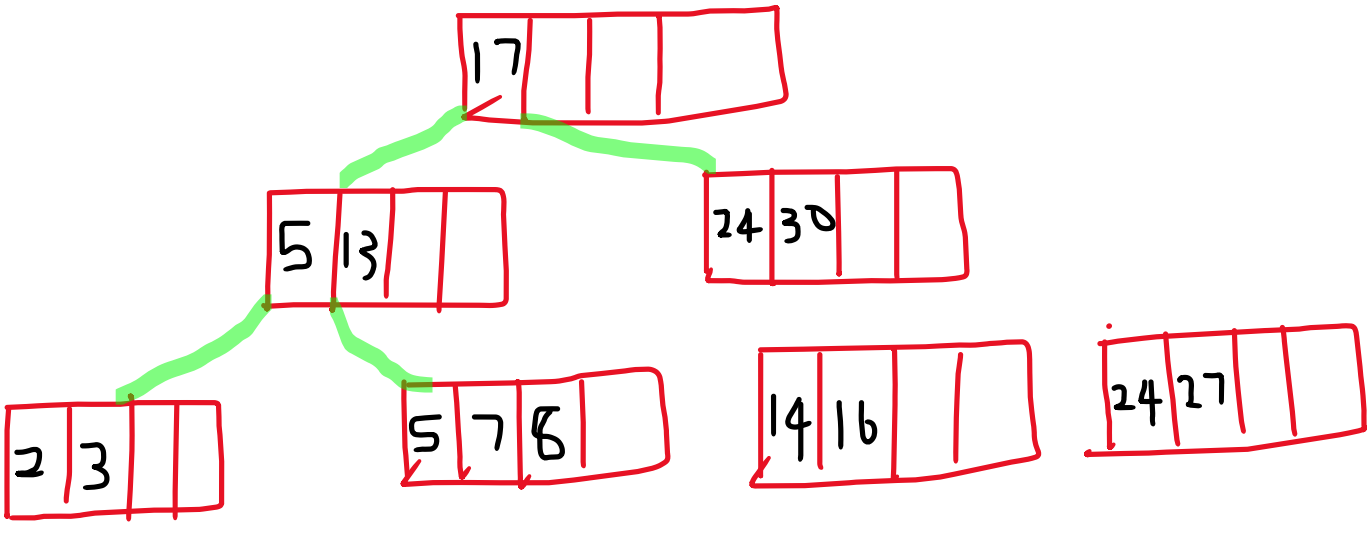


Hitann iht B+ tree ekt 8 insert krnn oni kiyla. Etkota meke venne me vge seen ekak. 8 pallehatta conditions valata anuva availla place vena (parent t vada kuda nm left and equal or vishalanm right). Ehidi 8 enne (2,3,5,7) node ekt. But ema node eke enough space ekk na 8 save krgnn. Assume ek 8 natuva 15 una nm (14,16) node ekt eka avill e deka madin place venva(14,15,16) lesa. Ey save una gmn sort venva. But ape scenario ekt anuva ek enne (2,3,5,7,8) but eke 8 enn space ekk na.

Ehidi krnne meka dekt bedala mada agaya aran eka **copy up** krnva. Etkota venne mema



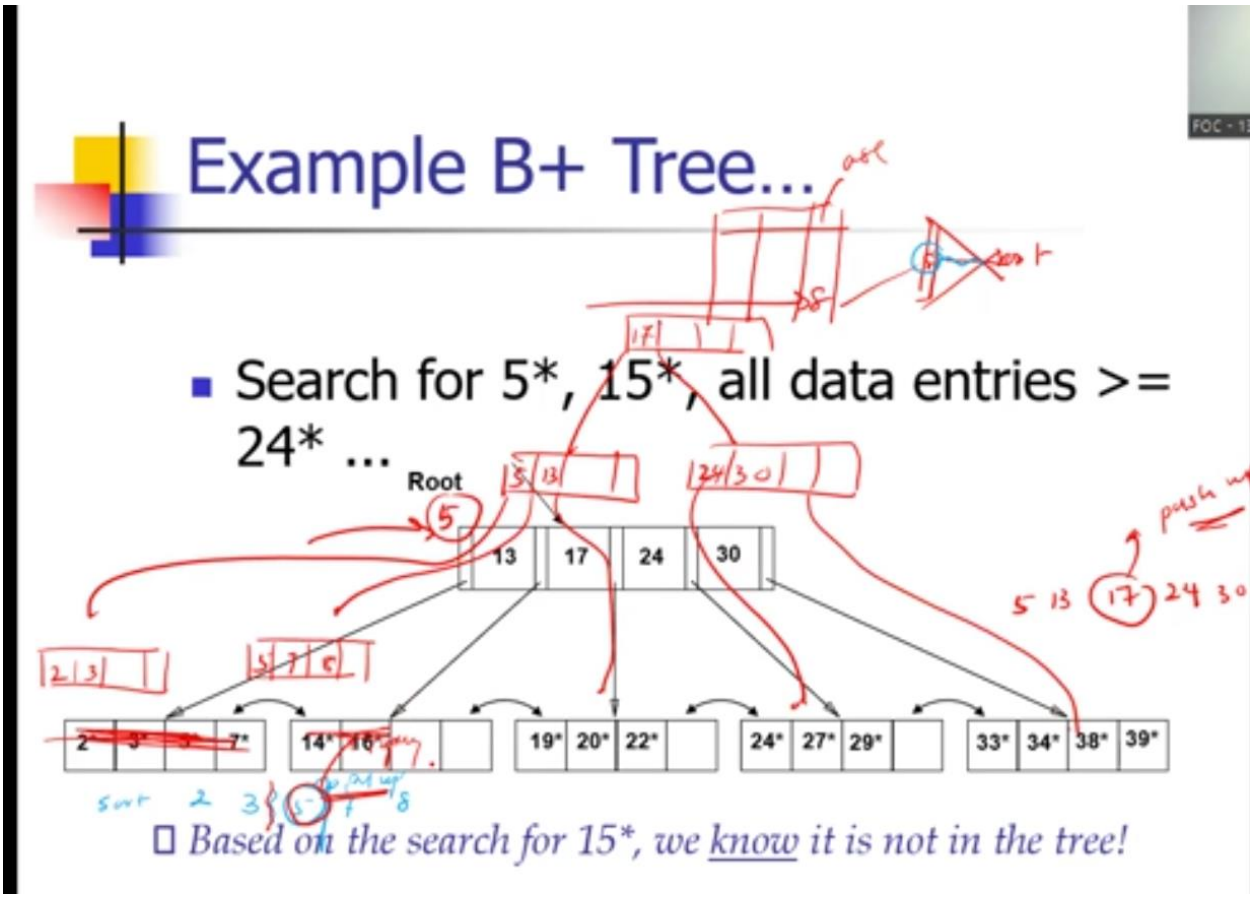
Passe 5 yanava udata. Parent ekt. Ehidi parent eke enough space ekk tiyeinm aulk na. but ema nattan ehidi venne **push up** krnva. Ehidi ape scenario ekt anuva etn space ek adui. Ek nisa value tika sort krl middle value ek push up krnva. Etkota ek tiyenne me vge



Push Up = Value ek remove krn udata ynva

Copy Up = Value ek etn tiyeddi copy ekk udata yavanava

Lecture note eka



Example For B+ tree

Student Table

Name	DOB	Age
Lahiru	2022/12/09	12
Sadun	2022/12/8	15
		11
		17
		20
		14
		19
		9
		13

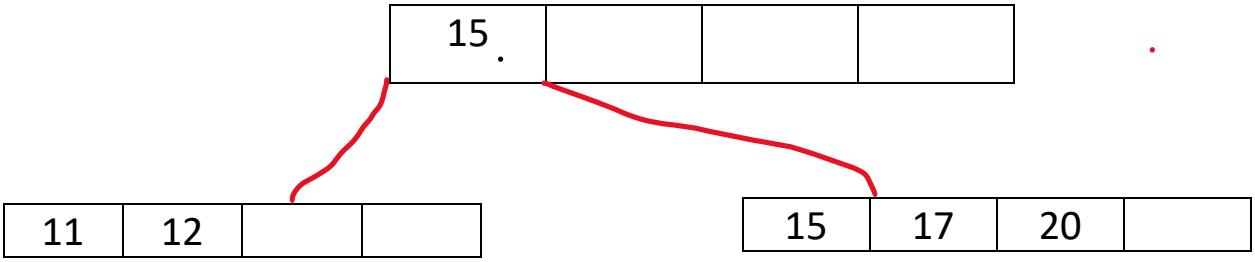
Then I'm going to create a B+ tree index for that table using the age column. Firstly, it will empty bcz table also empty. Then when we add data to table B+ tree index also fill it with that data. Assume api issellam Lahiru, sadun pilivelata data add krddi age avilla tiyenne 12,15,11,17 pilivelata. But ek table eke save vela tiyenne 11,12,15,17 vidiyt. Bcz kalin qwa vge index ekk data nitr save venne sorted vela.

(B+ tree ekk typical order number ek 100. It means one node can have 200 values. But in this case, we got 2 as an order number. So, one node can have 4 values)

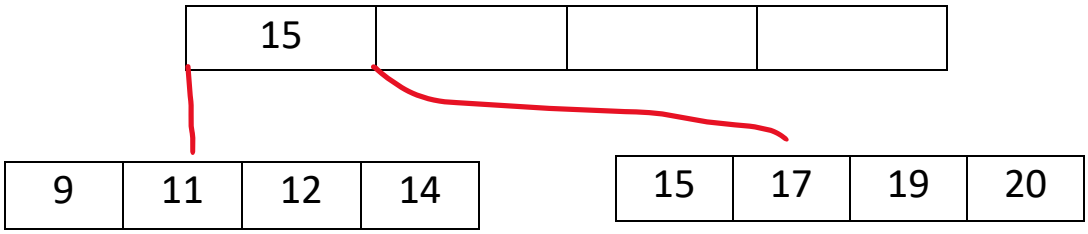
11	12	15	17

Then assume api passweni column ekat table ekt isert krddi e value ek(20) B+ tree index eket save venna oni. But first node ek full. Then api blmu ek krnne komada kiyla.

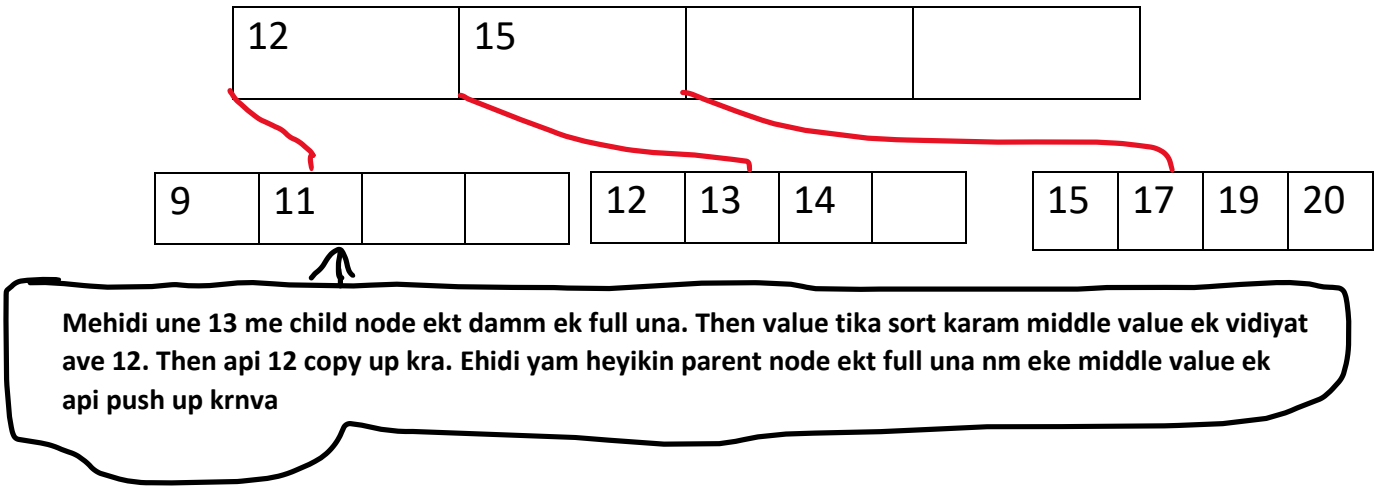
Issellam mekt 20 dala ek sorted krgnn oni. Etkota value tika enne 11,12,15,17,20 vidiyat. But apit ek node ekk tiyagann puluvn value 4k nisa ekk apita **copy up** krnn venva. Ehidi meke tiyn mada value ek api copy up krnva. Etkota diagram ek enne me vidiyt



Etana idn ilaga value tika add kr hati palleha tiynva. Then we will add 14, 19, 9, 12. **Hamatissema value ek add krl node ek sor krnva**

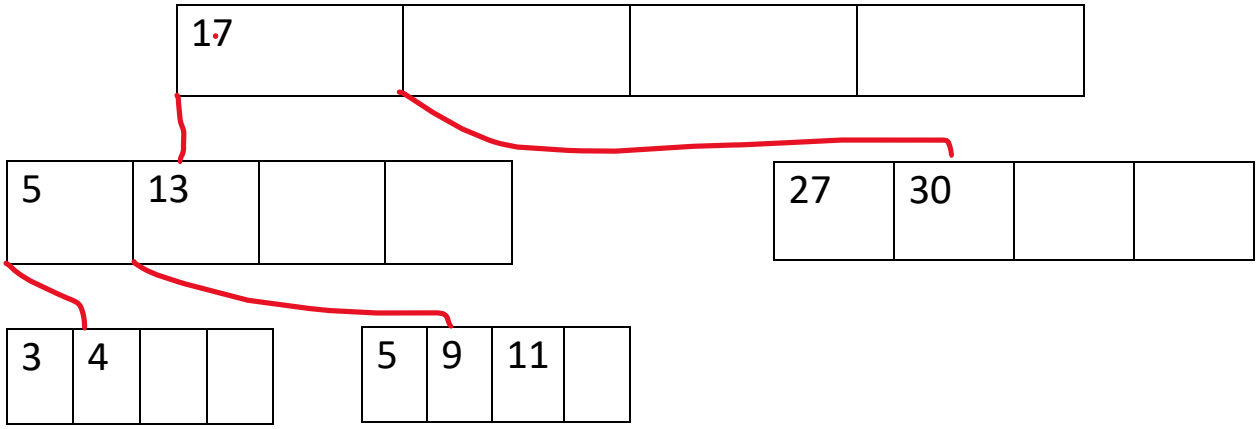


Finally, 13 add krn eka mm venam diagram ekkin pennannm,



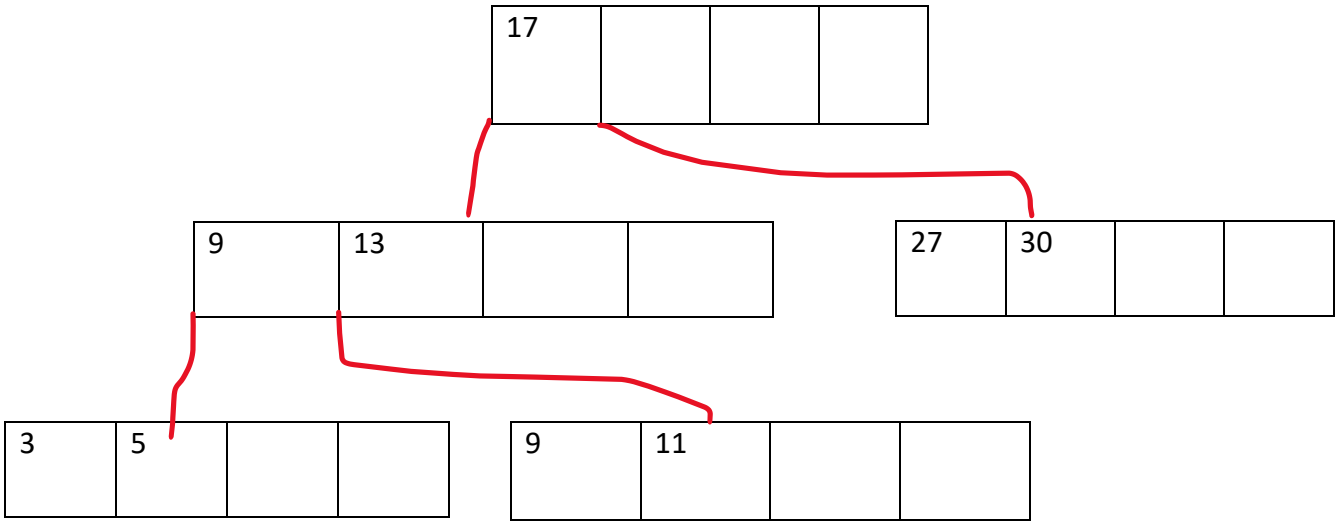
B+ tree Delete

B+ tree eke rule ekk tiyenva ek node ekk minimum 50% value tiyenna oni. E kiynne api gatta example vlt anuva ek node ekk aduma value 2kvt tiyenna oni. Root node ekk puluvn 50% vlt vada less than tiyagann. Anit evge ba. Root node ek kiynne parent node ekt.

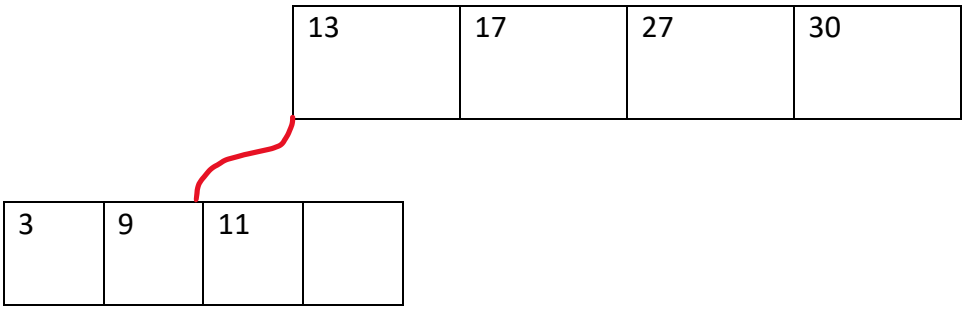


Mekt anuva apita 11 or 9 or 6 agayan valine ekak delete krnn unot its ok. Bcz etkota value 2k ituru venva. But apita 3 or 4n ekak delete krnn oni unot vade aul ynva. Etkota 50% na e node eke.

Assume 4 delete krnva. Then e vge velavata api krnne 4 delete krl anit pattat damage ekk venne nati venna ehen value ekk gena eka. After finalizing krnva. Etkota e node deka enne memai



Tava aulk tiyenva meke assume apita me eke 5 ain krnn oni. Etkota apita ekt anit node eke help ek gannt ba. Bcz eket dn tiyenne 50%. Then like that situation ekk api krnne e node deka merge krna eka Ehidi tibba node deka delete krl alut ekk tama hadenne. Itin tibb node deka delete una nisa e parent node ekt child node nati nisa e parent node ek api remove krnva. Ape scenario ekt anuva 9. The etnat aulk tiyei e parent node eket 50% data na. Then api aaai krnne e parent node dekat akatu krna eka. Ehidida tibba node deka delete vela alut ekak hadenva. Itin etota Root node ektd child la natin isa ekt ara merge krpu alut node ekt danava. Then ek mema avasanat enne



Hashing Data Structure

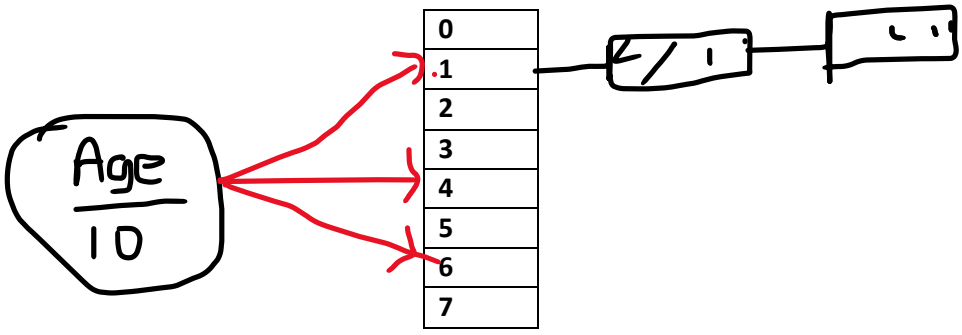
Hash-based indexes are best for equality selections. Can't support a range of searches.

B+ tree indexes are best for range selection.

Hashing pradan kotas dekai. Static and Dynamic hashing. Dynamic Hashing tavat kotas 2i Leaner hashing ha Extendible Hashing

1. Static Hashing

Meke krnne function ekk use krla index ek organize krna eka. Me kiynne api file organization vala ata kara Hashed file organization ekamai. Different ek tama Hashed file organization ek kre Table eke records valata. But Static hashing ek krnne Index ekt. Meke tiyna aula tama assume 10-20 tama hugak data tiyenne. Ehidi e block ek full unot ey rupe tiyn vidiyt tava block hadagen connect venva. E kiyenne ek long overflow chains ekk hdnva. Then performance ek down venva



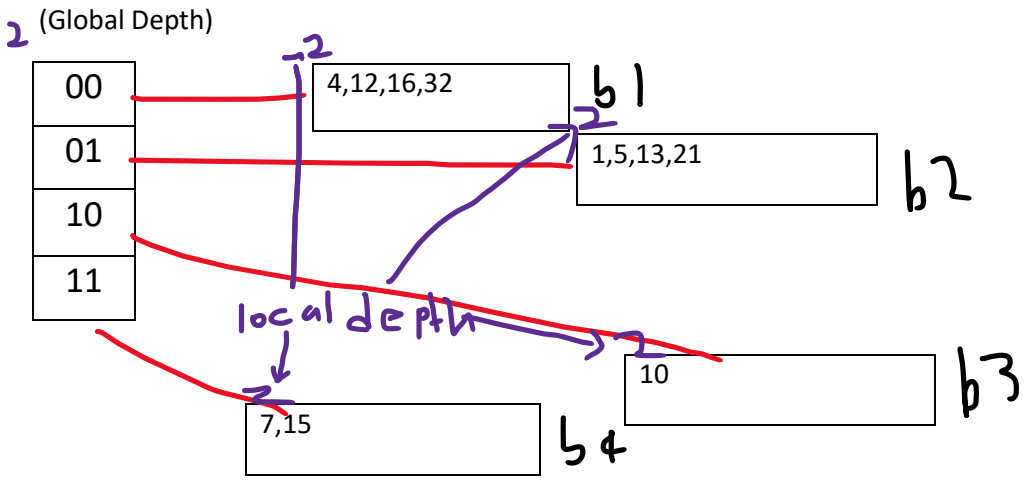
Problems:

- Insertion can create long overflow chains can develop and degrade performance.
- Deletion may waste space

Extendible and Linear Hashing: Dynamic techniques to fix this problem. (Me problem valat solutuin aran tama dynamic hashing ek avilla tiyenne).

2.1 Extendible Hashing

Meke venne me vge seen ekak. Global depth ekt adalav binary data combination ek hadagena ita adalava bucket tikata data dana ek mehidi karai. Assume Global depth eka 2i. Then Combination 4k hadann puluvn

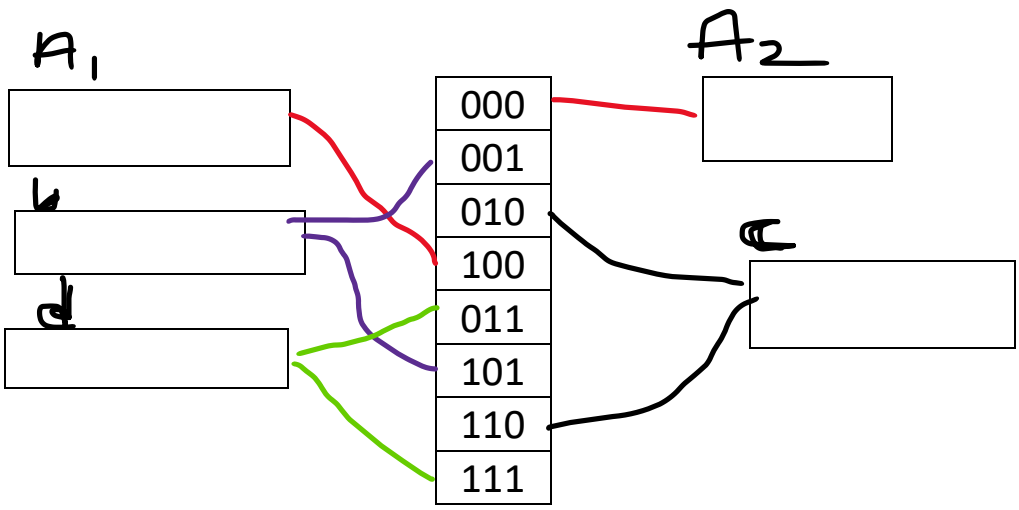


Me bucket valat value dala tiyn vidiya diha baluvot penva 4 – 0100 , 12 – 1100 vge avasana bit dekama 0 eva palaveni ekatat 10 – 1010 avasana bit ek 10 eva 3 bucket ektat lesa dala tiynne. Mehidi apita 11 insert krnn oni nm ehi binary agaya 1011 nisa api eka danne bucket 4 valata. Me vidiyt binary agaye last digit deka consider krmin value bucket valat damai.

Then assume apita age=13 studentge details gann oni nm krnne 13 binary vidiyt liyala avasan bit 2 anuva bucket ek tiranaya krl data gann ek. The meka equaling searching ekt patta speed.

Meket Sahamara velavat insert ekedi aulk enva. Assume apita 20 insert krnn oni. Then ek yann oni bucket 1 ekt but assume ek full. Then what we should do,

Ehidi api krnne bucket 1 ek tavat bucket ekkin extend krnva. E kiynne B11 and B12 kiyla bucket deck hadal ekt danava. Ema bucket ekk extend krddi Bucket eke local depth ekt ekk ekatu venva. ape scenario ekt anuva local depth ek 3k venva. But Meke rule ekk tiynva nitaram **Local depth ek Global depth ekt vada kuda ho smana venn oni kiyla.** Den mehidi ek violate venva. Then api ek fix krnne Global depth ekt ekkin ihala yvanal. Etkota venne mehemai,

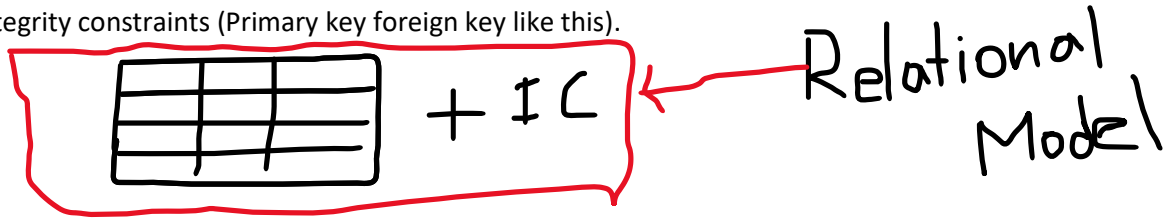


Meket baluvama terenva last two bit 00 dekak tiyen eva A bucket ekt, 01 eva b ekt vge ara piliveltama tama connect krl tiyenne

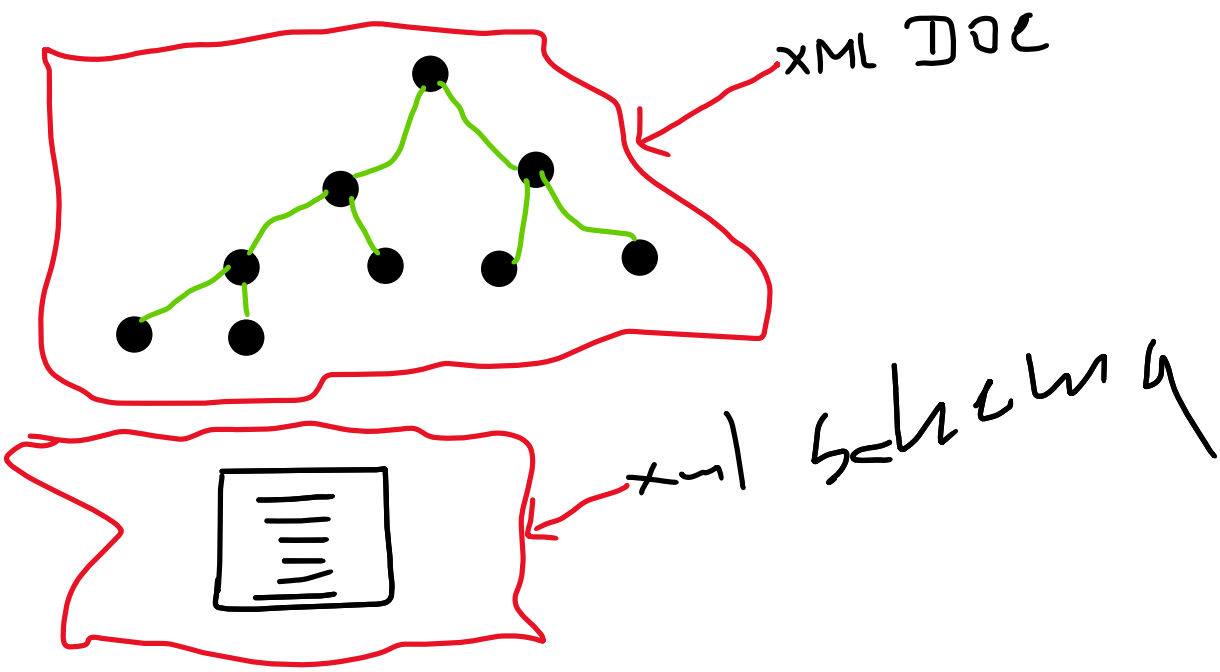
Meke deletion ekk krnn oni qwa. Ikm oluve tiyagann qwa dynamic evge insert ekedi bucket span venva vgema delete ekedi space ek aduvenva. Space waste venne na static hashing vge

Relational model vs XML Data model

The relational model's main structure is a relation. Relation is rows and columns. And there have Integrity constraints (Primary key foreign key like this).



In the XML model basic structure is a tree structure.



Xml data model ekk file dekak tiyei. Xml document and xml schema. XML schema also a document. Relational model eke IC(Integrity constraint) ek data validate krnva vge XML model eke XML schema eka xml doc ek data correctd nadd yann check krai.

Simply XML Schema is check whether our XML Doc data is correct or not.

within sql query, passer use **QUERY** function to execute Path expressions AND XQuery expressions.

Assume AdminDocs kiyla table ekk tiyei eke id ek ha product kiyla column dekak tiyei, product kiynne xml column ekk vana atara em table ekt data save krn hati pahata dakva ata. Meke tiyna anit pdf eke full sample code ekk tiyei

Insert into AdminDocs values (1,”

```
<catalog>
  <product dept="WMN">
    <number>557</number>
    <name language="en">Fleece Pullover</name>
    <colorChoices>navy black</colorChoices>
  </product>
  <product dept="ACC">
    <number>563</number>
    <name language="en">Floppy Sun Hat</name>
  </product>
  <product dept="ACC">
    <number>443</number>
    <name language="en">Deluxe Travel Bag</name>
  </product>
  <product dept="MEN">
    <number>784</number>
    <name language="en">Cotton Dress Shirt</name>
    <colorChoices>white gray</colorChoices>
    <desc>Our <i>favorite</i> shirt!</desc>
  </product>
</catalog>
```

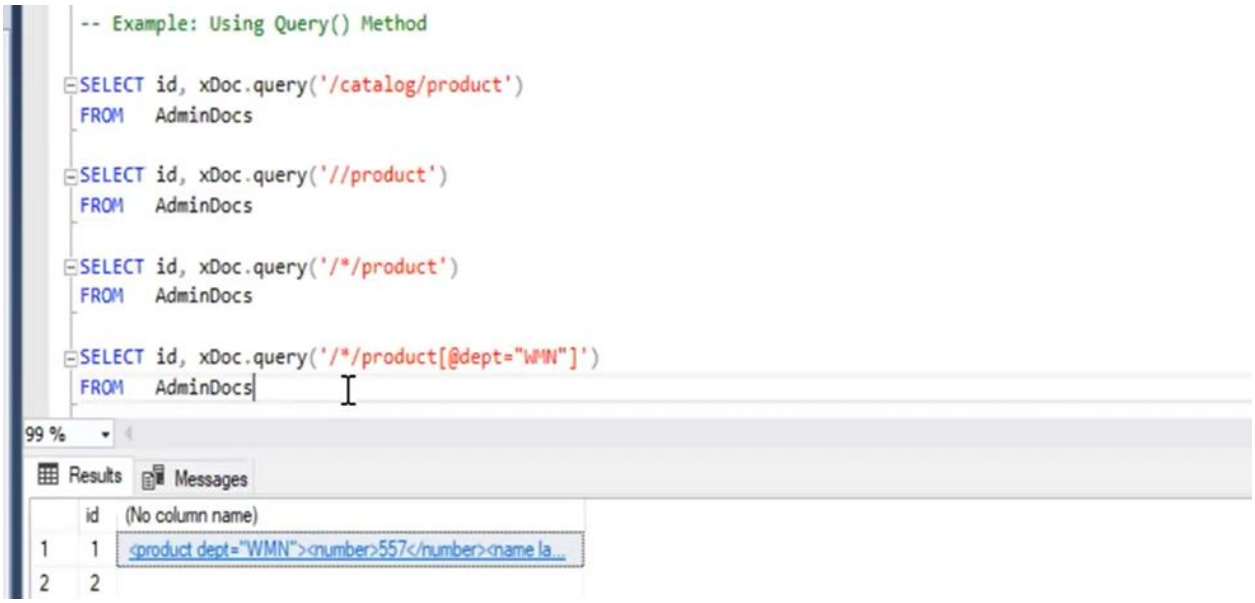
“);

Path expressions

select id, xDoc.**query**('/catalog/product')

from table name

1. **‘/cataolog/product’** - print all the product
2. **‘// product’** - tell going down the xml document and find the product tag and retrieve it
3. **‘/*/product ‘** - root tag can be anything but 2nd tag must be product tag and retrieve it * call wild card
4. **‘/*/product[@dept=” win”]’ -** this query has predicate condition **dep** his attribute of **product** tag



Axes

'/child::BAR[@name="JoesBar"]'

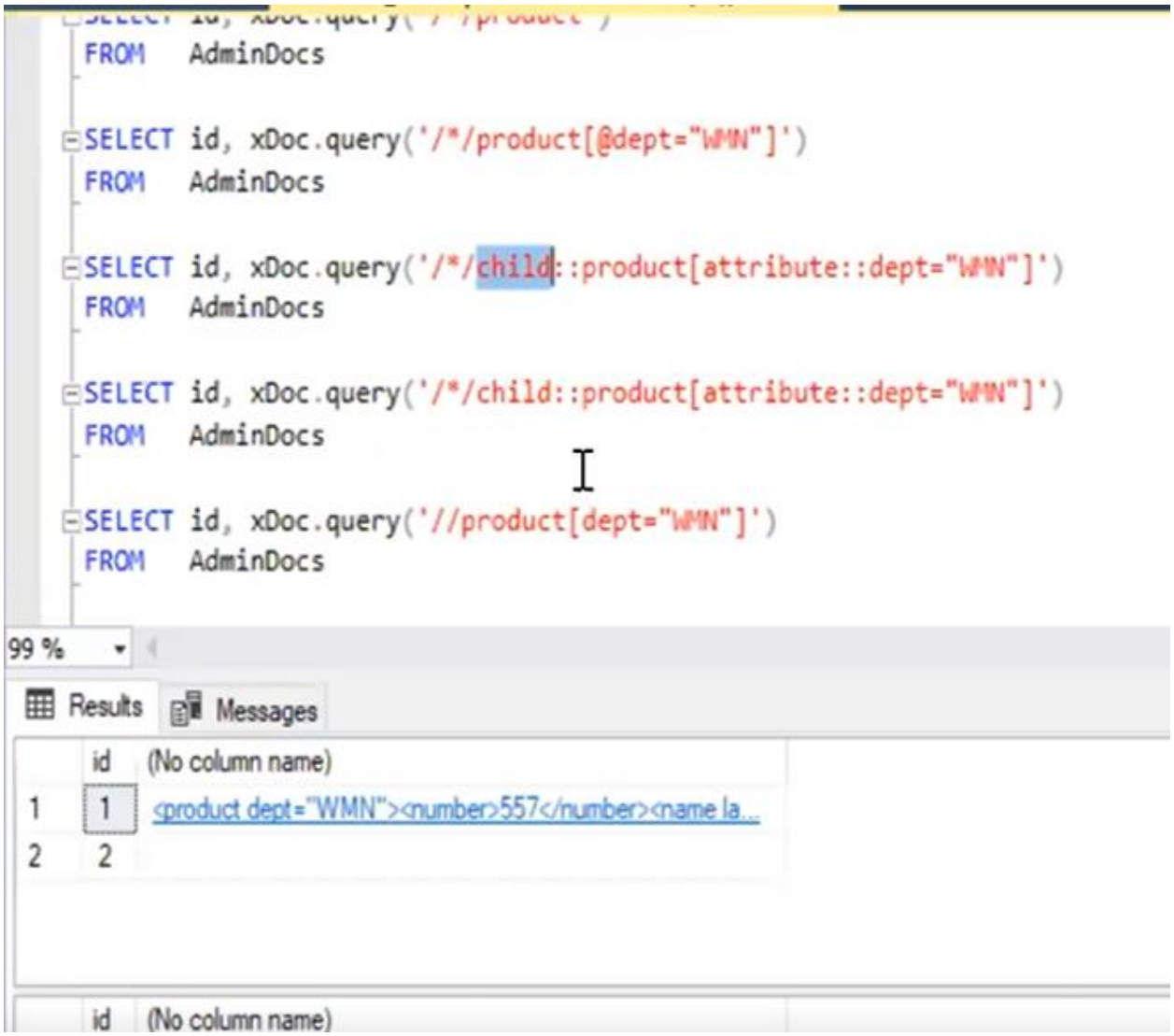
Axis Node test Predicate

- The axis (Optional) –
  - Direction to navigate (parent or child) default is child
- The node test –
  - The node of interest by name (child tag name)
- Predicate –

- The criteria used to filter nodes (used to filter already selected child attribute value)

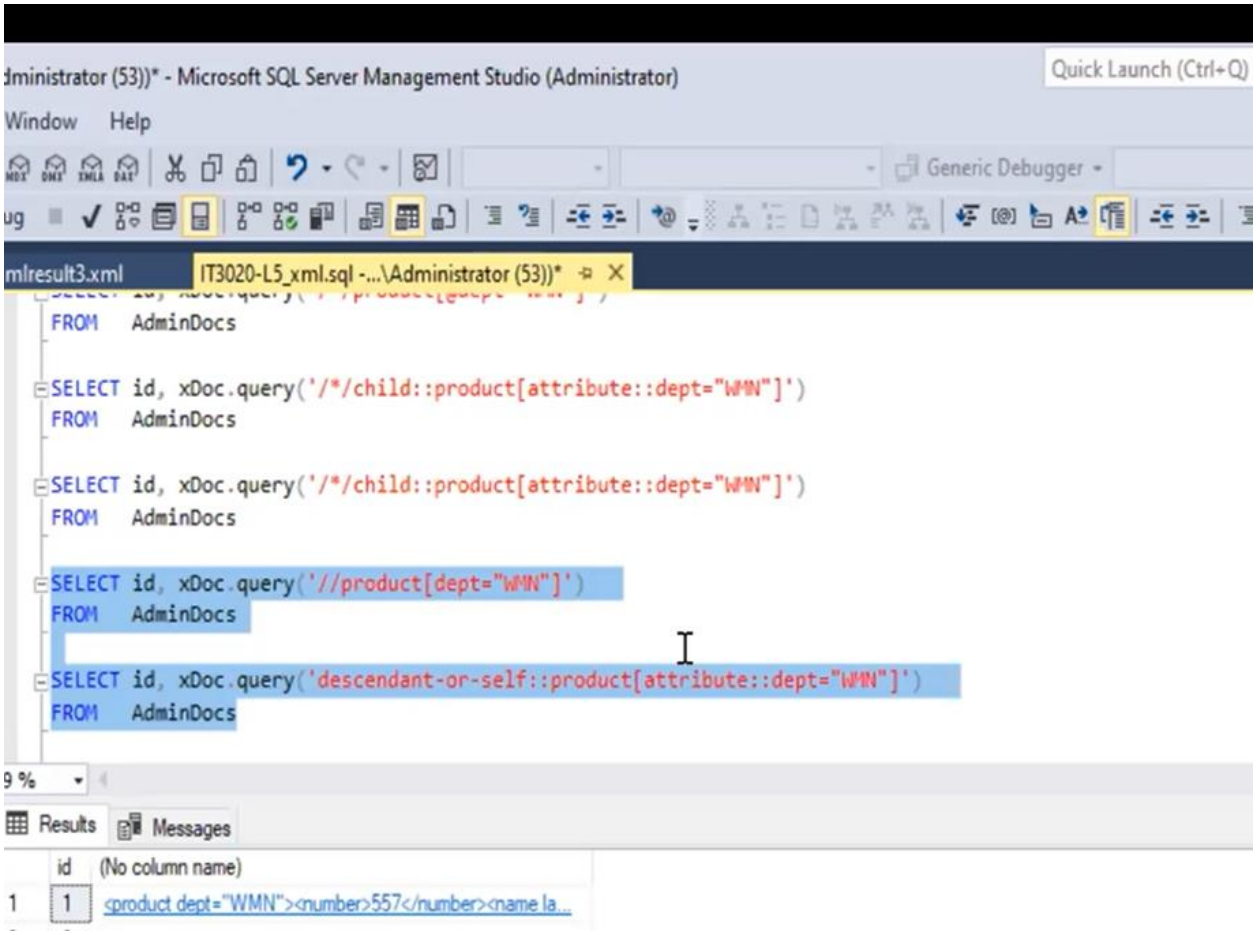
@dept=”wn” **same** attribute::dept=”wn”

5 ‘/*/product[@dept=”win”]’ **same** ‘/*/child::product[attribute::dept=”wvm”]’



// **same** descendent- or- self::

‘//product[dept=”wm” **same** descendent- or- self::product[attribute::dept=”wm”]



‘//product[number>500]’

Retrieve all the product number grater than 500 meke child tag eka athulata yane nathuva main tag eka retrieve karanava

‘//product/number[.gt 500]’

Check . current node grater than 500 print only the number

select id , xDoc.**query**('/catalog/product[2]')

from admindocs

where id=1

this will print product no 2

XQuery expressions

In xquery we have clauses

Ex: for, let, order by, group by, where, return this clauses call **FLWOR expression**

F - for

L- let

W - where

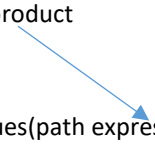
O – order by

R- return

For

Specify number of iteration you need

/catalog/product



For $i in list of values(path expression use to create list of values.one value become one element)

Let

Let use to declare a variables

Where

Use to mention conditions to be apply data

Order by

Use to sort to data

Return

Use to print values

Ex 1:

```
Select xDoc.query('for $prod in //product
                   Let $x:=$prod/number    same       ( //product/number   $prod=product)
                   Return $x')
From admindocs
Where id=1
```

Ex 2:

```
Select xDoc.query('for $prod in //product
                    Let $x:=$prod/number    same      ( //product/number   $prod=product)
                     Where $x >500
                     Return $x')
From admindocs
Where id=1
```

Ex 3:

```
Select xDoc.query('for $prod in //product
                    Let $x:=$prod/number    same      ( //product/number   $prod=product)
                     Return (<item> {$x} </item>)')
From admindocs
Where id=1
```

Add new item tag to every number tag modify the out put adding tags

Ex 4:

```
Select xDoc.query('for $prod in //product
                    Let $x:=$prod/number    same      ( //product/number   $prod=product)
                    Where $x >500
                    Return (<item> {data($x)} </item>)')
From admindocs
Where id=1
```

Remove existing XML tags and add our own tags (remove number tag and add item tags to the values)

Ex 5:

```
Select xDoc.query('for $prod in //product
                    Let $x:=$prod/number    same       ( //product/number   $prod=product)
                  Return if ($x>500)
                       Then <book>{data($x)}</book>
                        Else <paper>{data($x)}</paper>')
From admindocs
Where id=1
```

After checking return add if eles conditions to adding own tags and removing existing tags

----------Example using the exist() method

Ex 1:

```
Select id
From admindocs
Where xDoc.exist ('/doc[@is=123]')=1
```

Check xdoc column exist doc tag attribute 123 and if it is exist then exist function return 1 then relevant id is printed

How to modify xml document

That means how to adding branches to XML or how to remove branches from the XML

Add new element to the XML

```
Update admindocs
SET xDoc.modify('
Insert
 <section num=='2'>
        <title> backgroung </title>
</section>
After (/doc//section[@num=1]) [1]
')
```

After section no 1 insert section 2 to the xml

Delete existing element from the XML

```
update  admindocs SET xDoc.modify(‘
delete
  //section[@num=”2”]
‘)
```

Query Processing kiynne api data ekk retrieve krnn query ekk gahuvama ek database enginerun venne komada kiyla bln ekai.

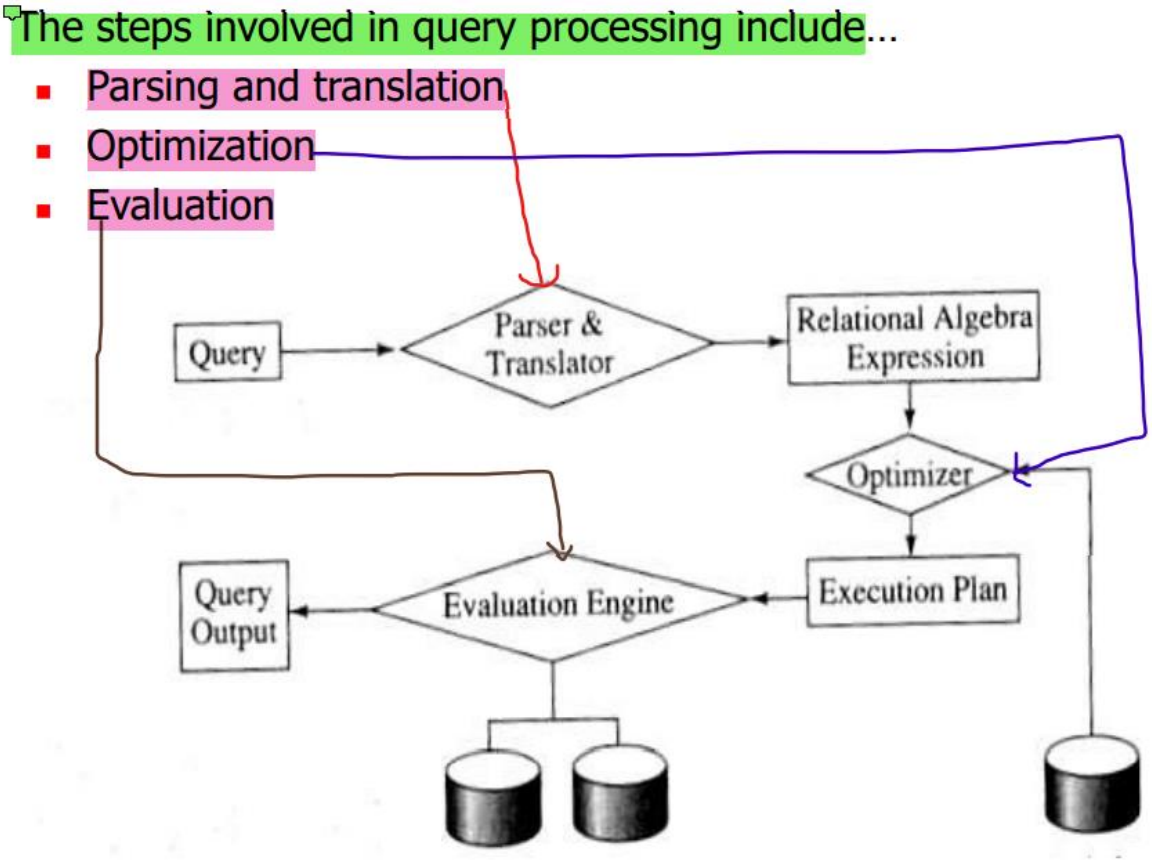
query-> Relational Algebra Expression->Execution Plan->Output

Eka sidda venne memai

Query -> Relational Algebra expression (Mekata use krnne parse and translator use krl)

Relational Algebra expression -> Execution plan (Meka krnne optimizer use krla)

Execution plan -> output (Meka krnne Evaluation engine use krla)

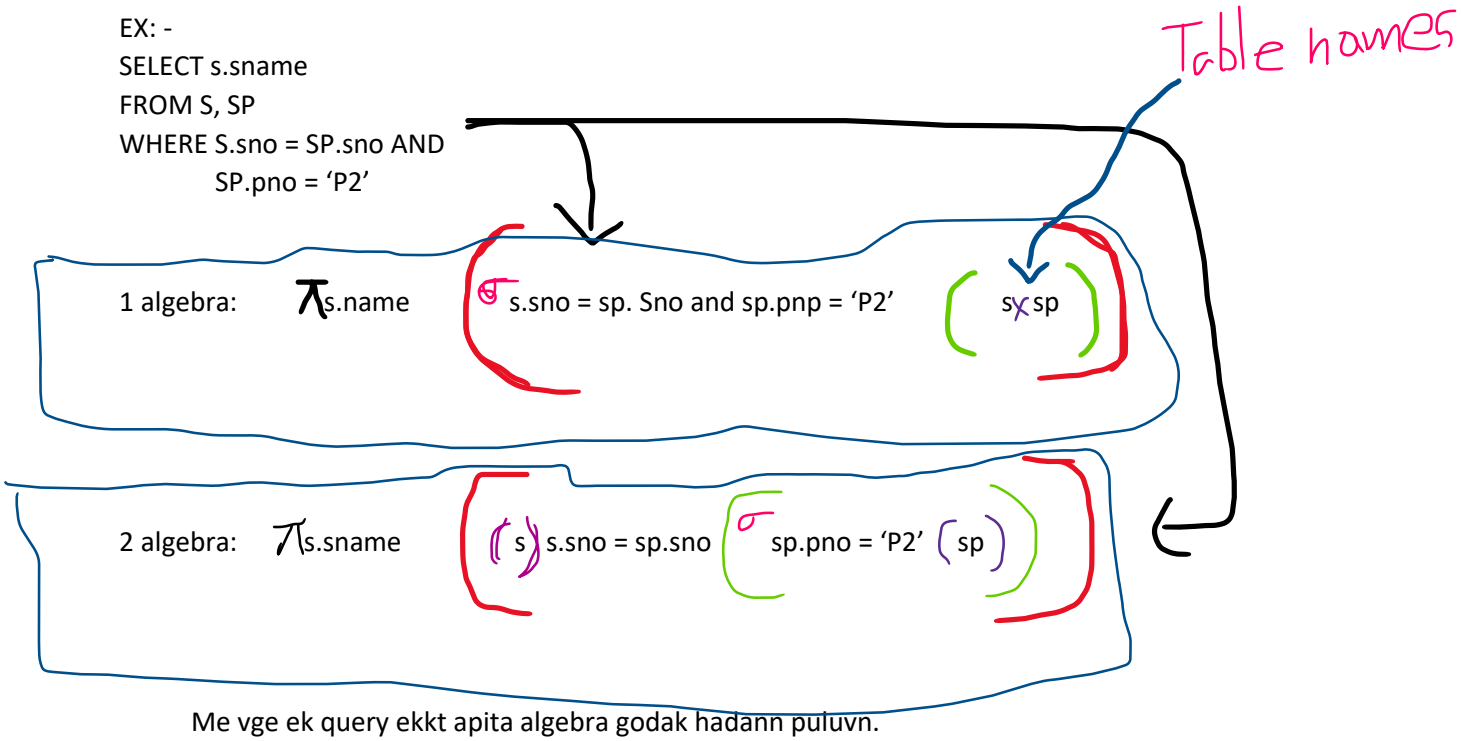


1. Parse and translator

Query ekk Relational algebra expression ekkt convert krnn tama meka use krnne.

EX: -

```
SELECT s.sname
FROM S, SP
WHERE S.sno = SP.sno AND
        SP.pno = 'P2'
```



Me vge ek query ekkt apita algebra godak hadann puluvn.

Meva Algebra operators ve

Me mark eken kiynne **join the table** krnn kiyn ek->

Me select operation icon eken kiynne **apply the condition** ->

Me icon eken kiynne **take out the result** kiyn eka ->

2. Optimization

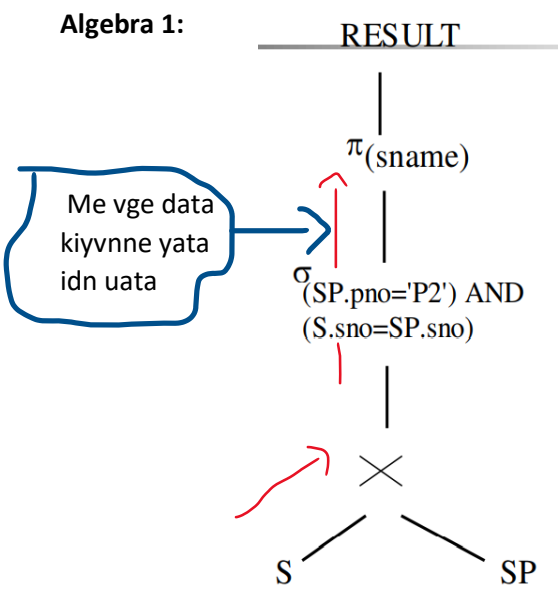
Meke krnne ihata hadagattu algebra vlin best execution plan ek (Hodama eka) toragann eka. Based on the cost we can decide what is the best one. Then me cost ek find krn hati api pahalin blmu.

Attatam database serves ekt time ek yanne hard disk eken data memaory ekt adinn(reading data) ha memory ek data hard disk ekt write krnn(Writing data).

Then I/O cost high nm performance is down, and I/O cost is law performance is high

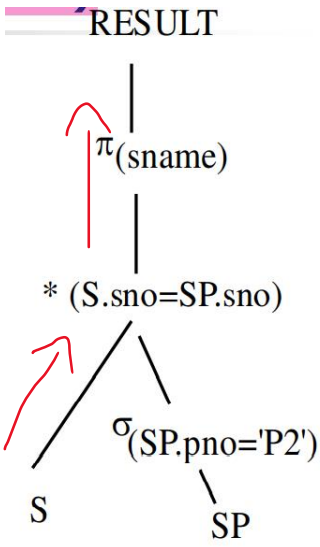
Ihata liypu algebra deke query tree ek pahat tiyenne. Mekt api **Query tree** or **Execution plan** kiyll kiynva

Algebra 1:



-

Algebra: 2



Dn api krnna oni me plan deken mkkd best ek kiyla bln eka.

Dn api blmu meke cost ek estimate krn hati. Assume

S (table) has 100 pages,

SP (table) has 10000 pages,

Algebra 1: Cost = 100*10000 = 1,000,000 read (Meke kiynne me vge seen ekk. Api assume kara S

1,000,000 write

1,000,000 read

Approximate Total cost = 3,000,000

table eke pages 100 ha SP table eke pages 100000 tiyei kiyla. Then Firstly apit hard disk eken data tika memory ekt adinn oni. eyt issellama harddisk eke data tika read krnna 1,000,000 cost ekk ynva and join krddi condition ek satisfied ev write krnn 1,000,000 cost ekk ynva Then mehi condition 2t tiyei ekt data aaai read krnn oni ekt tava 1,000,000 cost ekk ynva. Then ektuva 3,000,000)

Algebra 2: Approximate Total Cost = 10,000+100 = 10,100

Meke krnne issellam SP table eke data tika memro ekt gannav. Yet 10000 cost ekk ynva. E aragen ek join krnn kalin condition ekk blnva. Assume e condition ek satisfied krnne 50 record kiyla. Then e 50 record tika tama memry ek write krnne. Ek small amount ekk nisa api liynne na. meanwhile S table eke data tika adinva eke tiyenne pages 100. Then Total = 10100. Actually, meke enna oni 50*100. But api mehi ganne approximate cost ekk

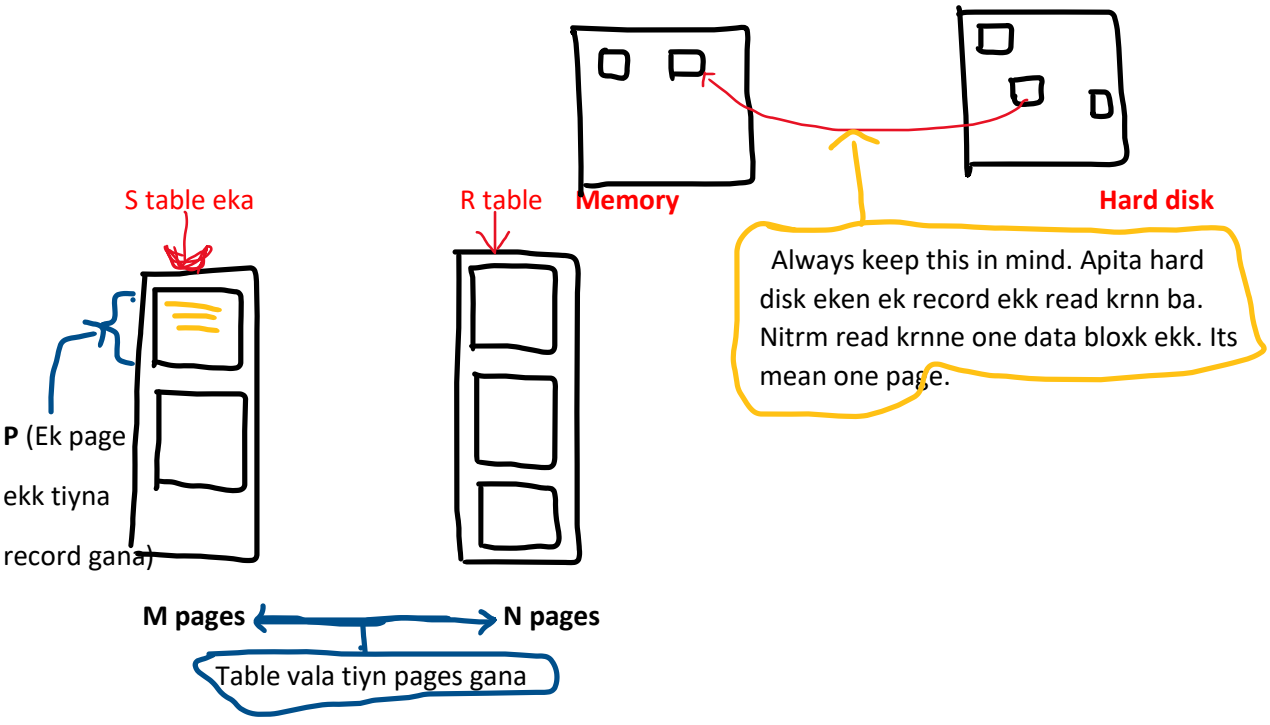
Then, best one is plan 2

Api metana idn kata krnne Joining algorithm kihipayak gana. Inbuild algorithm godak tiyenva. Evgen kihipayak api mehi kata krnva.

1. Simple Nested Loops Join

Meke venne Assume S and R kiyla table deck tiyenava. Then apita ganna oni s.id = r.aid record tika. Ehidi mema join algorithm ek krnne Firstly take the first record of R and check with S table records. The if the condition is satisfied join them. Then aaai R table eke ilaga ek aran S eke okkoma record ekk check krnva. Mema R table eke records okkoma ekin ek aran S table record ekk check krl ek satisfied nm join krnva. Meka thama me Simple nested loops join algorithm eke venne. Then api blmu meke cost ek komada calculate krnne kiyla.

Api meke assume krnva 1 page ekk = 1 disk ekk kiyla



Cost SNLJ (Single Nested Loops Join) = M + (N * (M*P))

Mehi S eke hama record ekkm aran R ekt ekk match karanava. Then S pages gana gannavan. S page eke gann hama record ekk ganatam R ek fully read krnva. Then S table eke tiyen record gana = M*P. Eke ganatam R table ek pages gana read venva. Then R table ek read vena vara gana = N*(M*P)

Assume Ihata table vala s table eke 5000 pages ha R table eke 20000 page tiyei. And S table eke 50 ek page ekk 50 tuples tiyei. Then,

Cost SNLJ = 5000 + (20000 * (5000 * 50)))

2. Page Oriented Nested Loop Join

Simple nested loop join ekei Page Oriented nested loop join ekei tiyn difference ek nm Simple ek krnne ek record ekk aran ek anit table eke record ekk check krn eka. But page oriented ek krnne single recoed ekk aran check krnva venuvata ek para one Page ekk tiyna okkoma record aran eva anit table ekt ekk compare krl join krna eka.

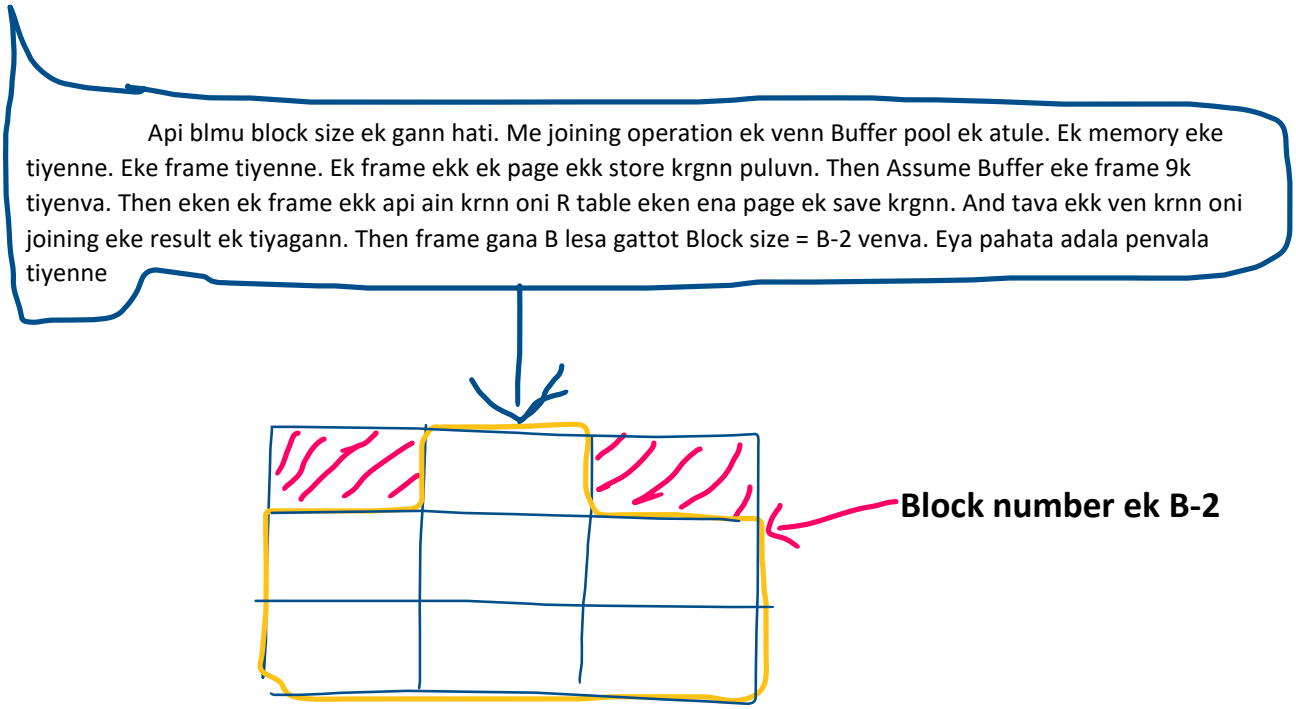
Cost PONL = M + (N * M)

Meke krl tiyenne S table eke page gana aran, Passe R table ek S table eke pages ganat read venva. (Bcz ek para ganne ek page ekk nisa.) Then apit R table ek read ven vara gana N*M lesa gann puluvn

3. Block Nested Loop Join

Block ekk kiynne ste of pages (page kihipayak) kiyn eka. Then mehi krnne Single record ekk Ho Single page ekk record okkoma gannav venuvata ekapara page kihipayak aran(It may be 5 pages) aran anit table ekt ekk check krl join krna ekai

Cost BNL = M + (N * Num of blocks in R table)

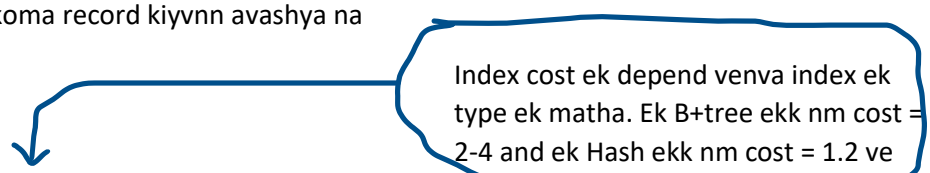


Then, Number of blocks = [M/ b-2]. Kotu varahan damme nattan enne decimal value. That's why put [].

Cost BNL = M + (N * [M/B-2])

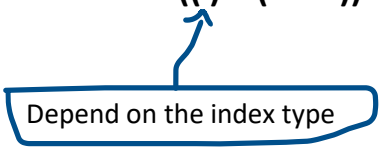
4. Index Nested Loop Join

Meke krnne mema seen ekk. Attatam api me venkm inner table eke record okkom kiyuveane. Ape scenario ekt anuva R table eke. But meke api okkoma record read krnne na. Assume Apita S and R join krnn oni me condition ek use krl. S.id = R.sid. Then api meke krnne R table eke tiyena sid column ekt index ekk hada gannav. Then assume api S table eke id = 10 record ek find krnva kiyla. Etkota venne api hadagattu index eke 10 tiyeida bll tiyei nm ekt adala record ek hoyagannav(rid use krl). Then apita mehidi inner table eke okkoma record kiyvnn avashya na



Cost INDJ = M + ((index cost) * (Num of record in S))

M + (( ) + (P*M))

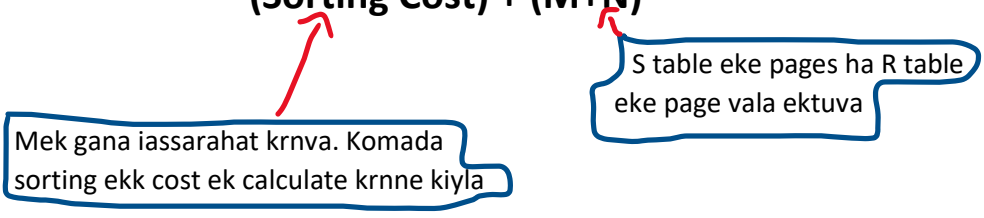


5. Sorting Merge Join

Meke sorting algorithm gana iassarahata krnva qwa. Meke saralavam venne assume apit S and R table join krgnn oni S.id = R.sid column use krl. Then mehi issellama krnne S table ek id column ekt anuva sort krgnn eka ha R table ek sid column ekt anuva join krgnn eka. Then passe check krl join krnva.

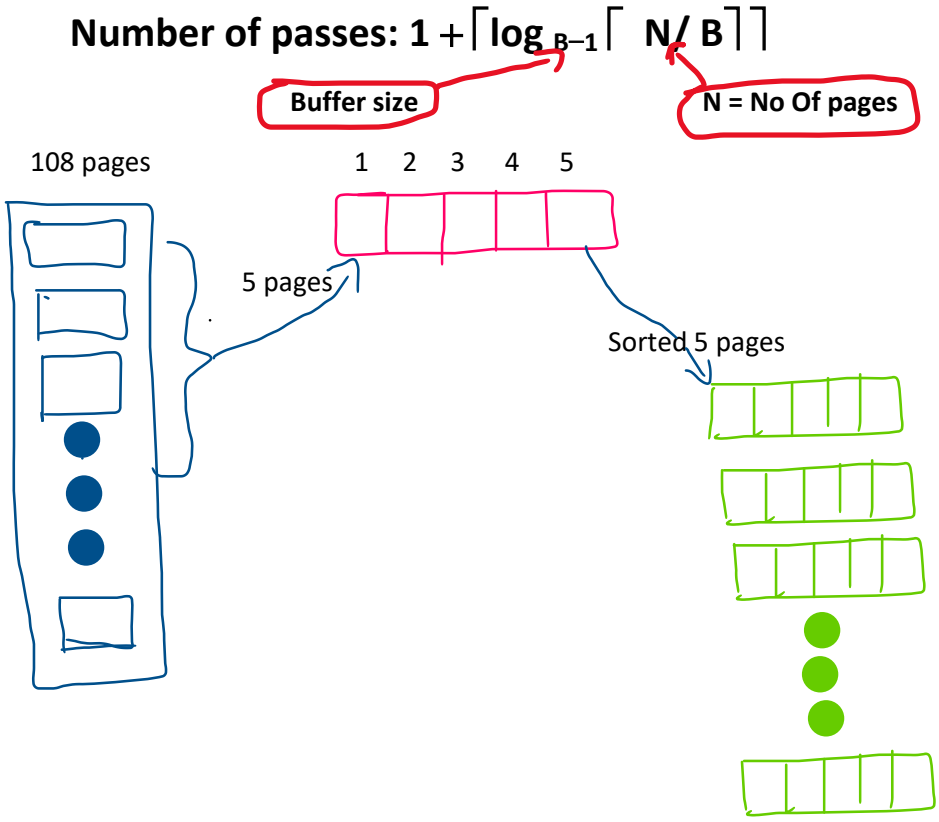
Cost SMJ = (Sorting Cost) + (Merging Cost)

(Sorting Cost) + (M+N)



5. External Merge Sort

Meke venne me vge seen ekak. Assume pahata table eke pages 108k tiyenva. Buffer eke size ek 5i. Buffer eke kiynne memory eke. E kiynne ek para yet read krnn puluvn page 5i. then apita table eken ek para page 5k aran tama sort krnn venne. Ema sort krpuva ven venam save krgnnava. Ape scenario ekt anuva ema new file 22k hadenva ([108/2] = 22). Then mehi venam venm haduna file tikat apit ekt krl thani sorted ekk hadann opni. Ek krnne step kihipayakin. Evata kiynne passes kiyla. Then meka sampurna sorted krnn yana passes gana me suthraya bavitayen hoyanna puluvn **1 +⌈log $_{B-1}$⌈ N/ B⌉⌉**



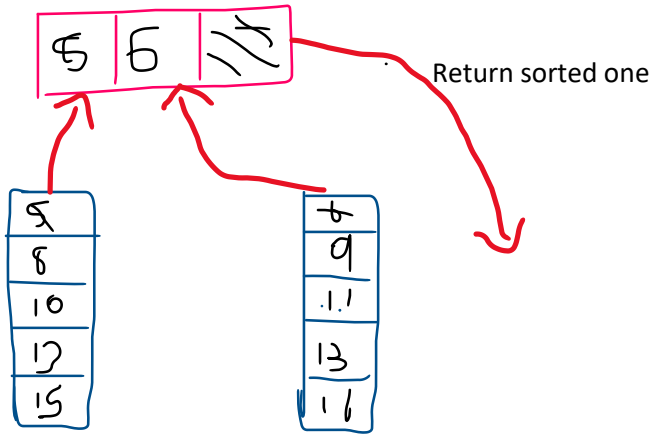
Pass 0 :⌈ 108/5⌉ = 22

Pass 0 vala krnne ihata rupe dakvena part ek. It means sorting 108 pages 5 at a time. At the end of pass 0, we will get separately sorted file 22k. (last file eke tiyenne pages 3i)

Pass 1 (also called B-1 way merging)

Meke krnne meka. Dn attam apita file 22k tiyenva. Then api 1 sorted ekkne hada gann oni. Itin mehi krnne me file join krna ek. Ek para file kiyk join krnvad kiyla danagnn tama B-1 dila tiyenne. E kiynne Buffer size eken 1k adu karama en ganak ek para join krnva. Ape scenario ekt anuva 5-1 = 4k.

Ek hadeenne mehemai. File merge krddi buffer frame eken ek buffer ekk apita ain krnn venva sorted file ek return krnn. Then apita file 2k sort krnn oni nm apita size 3ka buffer frame ekk oni. File 3k ek para merge krnn size 4ka buffer frame ekk oni. Ey pahat rupayen vadat Vistara ve.



Then ape scenario ekt anuva pass 1 vala separately sorted file 4k api ek para merge krnva. Then the end of pass 1, we have 6 separate files. Ema file vala pages 20,20,20,20,20 and 8 lesa save ve.

Pass 2

Mekt eddi apita separated sorted file tiyenne 6i. Meke krnnet pass 1 eke krpu ekmi. Page 4 gane aran me file 6t api merge krnva. At the end of the pass 2 apita separately sorted file 2k hambneva. $\lceil 6/4 \rceil = 2$.

Pass 3

Mekt eddi apita separately sorted file 2i tiyenne. Then mekedi krnne kalin krpu ekamai. Attatam apita file 2k merge krnna buffer 3k tibbam ati. Then api mekt use krnne 3i. Then at the end of the pass, 3 we will have 1 separated file $\lceil 2/3 \rceil = 1$.

Cost EMS = (2 * No pages) * No of passes

Example 1:

File pages = 1000 and buffer 20 frames. Estimate the cost of the external merge sort.

Answer:

This one has 1000 pages, and the buffer size is 20. Obviously, this one can't load into computer memory. So, to sort this one we use an external merge sort

Pass 0 = $\lceil 1000/20 \rceil = 50$

Pass 1 = $\lceil 50/19 \rceil = 3$

Pass 2 = $\lceil 3/4 \rceil = 1$

Cost = (2*1000) * 3

= 6000 I/O

Example 2:

File pages 1500. 10 buffer frames.

Answer:

Pass 0 = $\lceil 1500/10 \rceil$ = 150

Pass 1 = $\lceil 150/9 \rceil$ = 17

Pass 2 = $\lceil 17/9 \rceil$ = 2

Pass 3 = $\lceil 2/3 \rceil$ = 1

Cost = (2*1500) * 4

= 1200 I/O

Transactions and Concurrency Control – Part 1

Transactions

Transaction means unit of work. It means the transaction is not single SQL command. EX: - Mm mge account eken Rs 100 k danava yaluvekt. Then issellam mge account eken 100 adu venna oni. So ekt update query ekk liynva mge account eken 100k aduvenna. But still transaction not complete. Bzc transaction ek complete venn nm yaluvge account ekt Rs 100k add venn oni. So ekt update query ekk liynva yaluvge account ekt salli add venn. Men me tika unam tama transaction ekk kiynne. That's why called unit off work

Each SQL command executes not a separate transaction. (Transaction kiynne single SQL command ekk nevei)

All SQL commands belong to one unit of work called a transaction. (Unit of work means ek karyak sampurna kirimata)

All SQL commands together generate a sequence of read writes in a database called a **transaction**.

Properties of transaction

- Atomicity

  Transactions mean a sequence of actions (read from DB, write to DB). Atomicity kiynne all of their actions happen or not happen. Vent akaretkt kiynvnm Atomicity partially done transactions are not allowed.

  Assume if DBMS allowed to partially done transaction, according to our previous example mge accpunt eken 100k aduvela ya;uvge account ekt add novi tiyenna puluvn

  Meke procedure ek nm all actions successfully complete unot commit action will happen. Commit means all modified data write to disk

  Abort means undo. So, all actions not succeefully complete the abort action will happen

  - The transaction is a sequence of read-write all actions happen, or none happen called atomicity.
  - For every transaction, either all actions within the transaction are carried out or none are.
  - A transaction might commit after completing all its actions
  - Every action that happens successfully will Commit otherwise aborted.
  - A transaction aborts after not completing all its actions successfully. (Transaction cancel)
  - DBMS logs all actions so that it can undo the actions of aborted transactions.

- Consistency

  Transaction ekk veddi if transaction violates any of the integrity constraint. Our database getting inconsistent data. Me velavat constancy property said for transaction will not violate any integrity constraint.
  As a developer, we set correctly all the integrity constraints database is responsible to maintain a database consistency property

  EX: - according to our previous example, account ekk aduma Rs 20k tiyenna oni kiyla constraint ekk dala tiynva Table ek create krddi. Then mge account eke tiyenne Rs100 and mm ek Tava kenekt send krnn haduvot mt e transaction ek krnn bari venva BCZ of database consistency. BCZ 100m dammot account eke 10k ituru venne na

  - Every transaction should see the correct data set.
  - The developer should implement all the integrity constraints (ICs), then the database is responsible for database consistency.
- Isolation

  App ekk DB ek usersla parelly access krnva ne. then mehidi conflict enn puluvn. Itin men me conflict handle krnna tama Isolation of transaction use krnne

  Ex: - Assume mm mge phone eken ha lap eken mge banking app ekt lofg vela innava. Mge account eke 1000k tiyein mm ekm velave phone eke Rs600k realod ekk dagann gmn lap eken Rs1000k tava kenekt yavann try krnva. Mek karnn bari venn oni ne. bcz account eke tiyenne 1000i mm 1600k vada krnn hadanava. Men me vge seen handle krnn tama me isolation use krnne

  - Isolate different transactions and handle conflicts also executing all transactions in some serial order.

- Durability

  This simply survives transactions from system crashes and failures. Network failure, memory dump, server failure, etc.

  - This property ensures that transactions survive or recover from system crashes and failures. (Power failure, network failure)

Transactions & Schedules

Sometimes DBMS is executed 100 transactions at parallelly

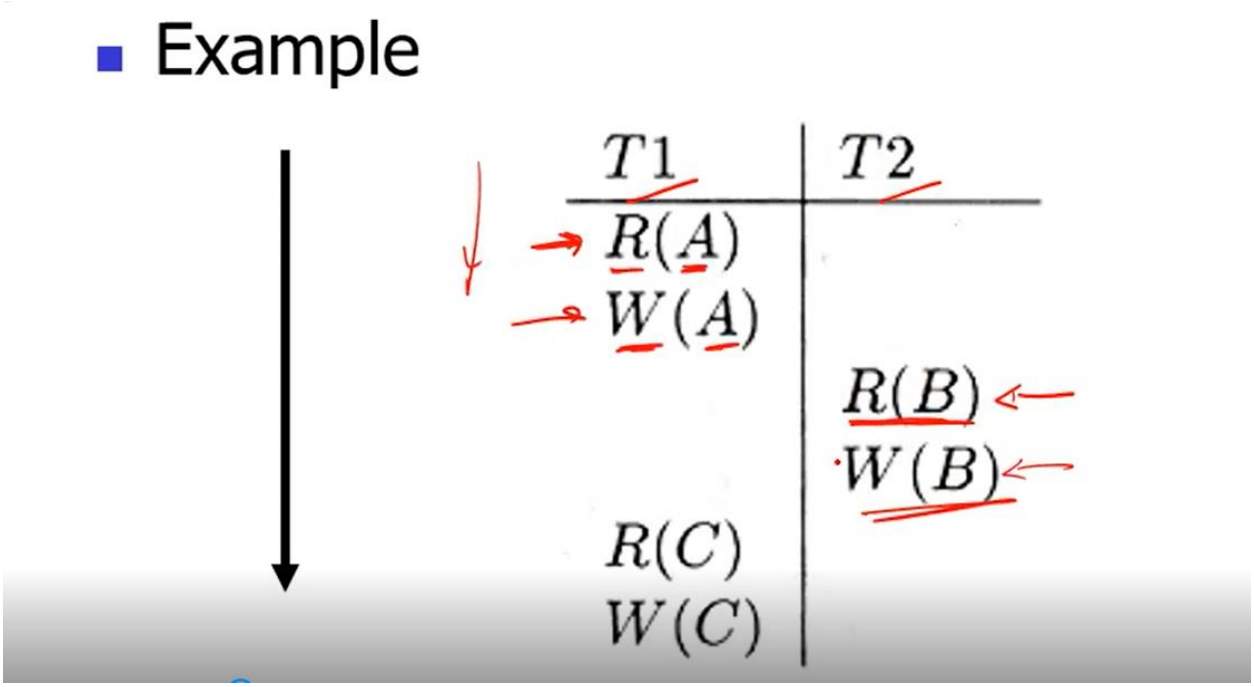
The transaction has 4 actions

1. Reads
2. Writes
3. Commits
4. Aborts

In schedule only schedule those 4 actions

Scheduling those actions is really important to avoid conflicts.

Meke kiynne me vge seen ekk. Pahat rupe diha balann. Eke T1 kiynne bank manager and T2 kiynne user kenek kiyla hitann. Dn Bank manager oni siyaluma userslge account balance check krnn. Assume ekt oaya 2k ynva. Itin meka vena ataratura use kenek eyge account eken Rs 1000k withraw krgnnva. Ekt vinadi 2i yanne. Itin dn me paya deka ek velavak process dekak venva. Ehidi conflict ekk enva. Itin ek maga arinn tama transaction scheduling hadunvl dila tiyenne. Attatm mehidi krnne user salli withdraw krddi bank managerge process ek navattal userge ek krla userge process ek ivara unama aaai bank managerge process ek krn eka.



T1 and T2 mean transactions.

R(A) mean → A object read by T1

W(B) mean →B object written by T2

Concurrent Execution of Transactions

CPU can process one transaction while another is waiting for a page to be read from disk

short transactions should give priority.

Interleaving actions of transaction mean jumping between transactions and executing actions.

Different types of scheduling

Serial schedule (This happen sequentially)

Saralavama serial schedule ek kiynne ekkt passe anit ek kiyna eka. Ex: - assume T1 and T2 kiyla transaction 2k tiyei. Then there are two possible serial schedules. T1 run first and after it finish t2 run. Mek ven venm sidda vena nisa no conflict. So, at the end getting correct data

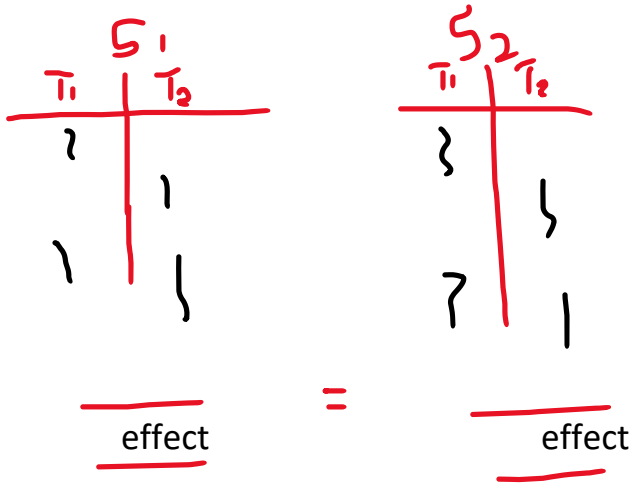
- one transaction followed by other transaction
- no clashes or conflicts between transactions
- **at the end getting correct data**

T1: Read(A)
T1: Write(A)
T2: Read(A)
T2: Write(A)

Equivalent schedules (This happen parallelly)

different schedules running in a different order but the effect of executing in the database is the same both of the schedules are Equivalent schedules

EX:-



T1: Read(A)
T2: Read(A)
T1: Write(A)
T2: Write(A)

meke kiynne schedule 2k different way valat execute krt ihata rupaye tiyena vidiyat At the end effect will be same. Effect mean result ek itherwise database ekt karpu change ek. Meke me S1 and S2 kiyla schedule deken krnne ekm de. Natuva vena vena Karyn dekak nevei. Ekm karya different way vlt krnne

Serializable schedule

Transaction ek parallelly (like equalent schedule) unata eken ena effect ek serial schedule ekk affect ekkt saman eva Seraializable schedule lesa hadunvnva

While doing transaction parallelly effect get from schedules equal to some serial schedule effect called Serializable schedule.

doing transaction parallelly also getting correct result at end of execution.

T1: Read(A)
T2: Read(A)
T2: Write(A)
T1: Write(A)

Serializability

Multiple transactions executed concurrently must be equivalent to some serial execution of the transactions in order to preserve consistency

The order of transactions within the schedule is not important

Anomalies with Interleaved Execution

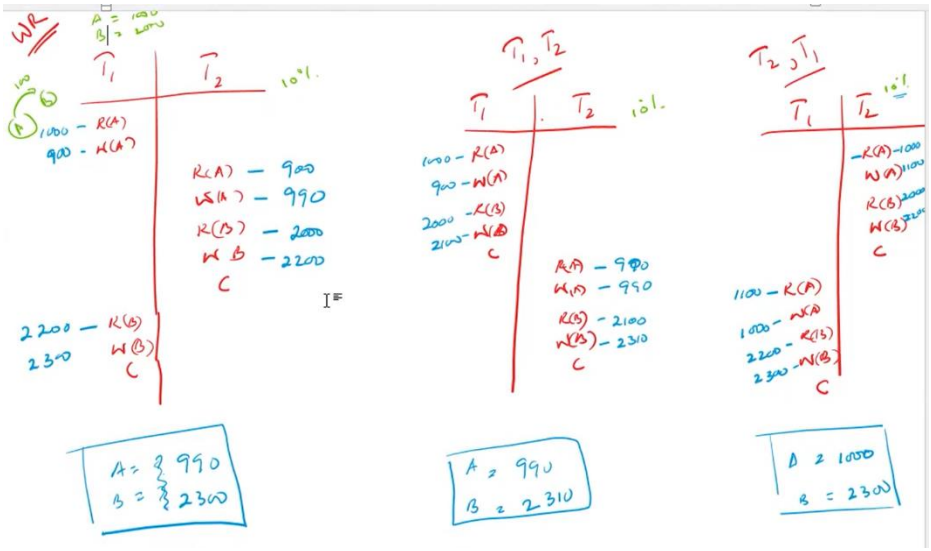
Meke kata krnne parallelly yaddi enn puluivn conflict gana. Pradana conflict 3i. e kiynne men me 3 gana. Assume t1 and t2 prallely run venne t1 mkkhri read krddi t2 t read krnvnm aulk na. but t1 read krddi t2 write krnvnm conflict ekk enva. E vgema t1 write krddi t2 read krnvnm et conflict ekk enva. E vgema dekm write krnvnm et conflct enva. At lease one write then there is a conflict

T_1	T_2	
R(A)	R(A)	no conflict
R(A)	W(A)	have conflict
W(A)	R(A)	have conflict
W(A)	W(A)	have conflict

So, menna me conflict 3 gana pahata kata krl tiyenne

1. **Reading Uncommitted Data (WR Conflicts, “dirty reads”)**

   Meke api kata krnne write and read conflict ek gana. Mekt api kiynva dirty read kiylt. Emek pahat rupe penvala tiyenne account A gen account B t Rs 100 transaction ekk krn seen ekk. Meke interest ek 10%. Its mean tiyana ganat 10% poliyak denva



- This is a not serializable schedule. Hetuva tama api palaveni ek concurrently run krl tiyenne 2 and 3 serial execution ekk krl tiyenne. So meka serializable venne nm palaveni eke result ekt ekk 2 ho 3 result samana venn oni. But meke ehema vela na. so meka serializable na

Before one transaction is committed another transaction read and writes uncommitted data will occur conflict. Kiynne palaveni eke 900 – W(A) kiyn process ek t1 patte vela ek commit krnn kalin read krnva t2 eken. So meka tama WR conflict ek. But blnn T2 ge W(B) – 2200 and t1 R(B)-2200 eke aulk na. bcz T1 T2 ge data kiyvnn kalin ek commit krl. Committed data reading is not a problem. But uncommitted data reading is problem

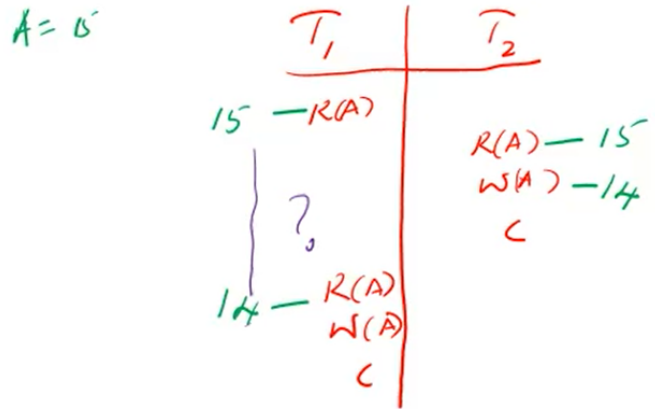
2. **Unrepeatable Reads (RW Conflicts)**

   Meke kata krnne Read Write Conflict ek gana

   Read the same value repeatedly when getting the inconsistent value, we call Unrepeatable Reads

T1:	R(A),	R(A), W(A), C
T2:	R(A), W(A), C	

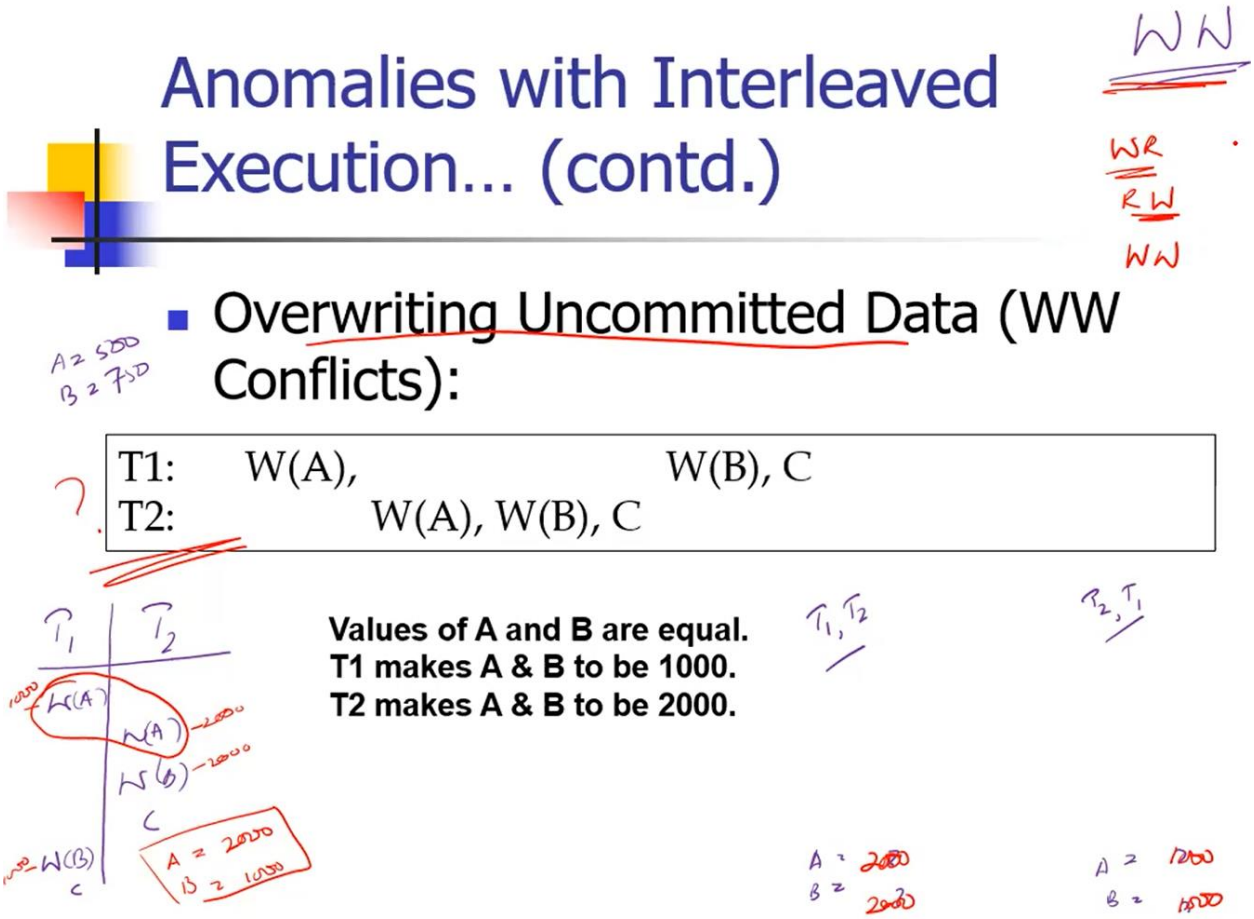
Example T1 is incrementing A by 1 and T2 is decrementing A by 1



Meke tiyena aula tama T1 vala blnn issellam value ek 15i. but kisima change ekk T1 patte venne natuvat dvenai para value ek change vela 14 kt. When we read some value and the same value read repeatedly get a consistent value

3. **Overwriting Uncommitted Data (WW Conflicts)**

Meke api kata krnne write write conflict ek gana



Meket tiyenne palaveni eke vge seen ekk. Concurrency ek effect ek serial execution eke affect ekat ekk saman na

Unserializability

The root cause for unserbalizable schedules (or violation of Isolation property) is conflicts.

- WR conflict -> **dirty reads**
- RW conflict -> **unrepeatable read**
- WW conflict -> **overriding uncommitted data**

  Me 3 nisa apit labenne unserializable schedule ekk**.** So api meva avoids krnn oni. Then apita labenne serializable schedule. It means can execute transaction parallelly while getting correct result

To avoid these conflicts, we can use concurrency control protocols

Then can get serializable schedules

Serializable schedules mean can execute transactions parallel while getting correct results

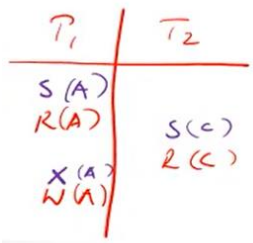
There are many types of concurrency control protocol

Lock-Based Concurrency Control

Strict Two-phase Locking (Strict 2PL) Protocol

It has 3 rules. Using these 3 rules we can avoid above 3 problems

1. Each Xact must obtain a **S (shared) lock** on object before reading, and an **X (exclusive) lock** on object before writing



2. All locks held by a transaction are released when the transaction completes (only released after committing). Meke kiynne changes commit krnkm transaction ek loack krl tiynva kiyla

3. If a Xact holds an X lock on an object, no other Xact can get a lock (S or X) on that object.

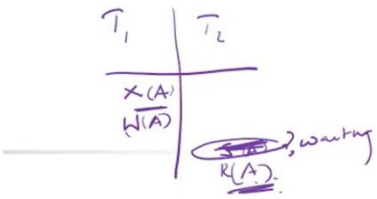
   Ex: for example, if T1 holds X lock of one object any other transaction can't hold S or X lock on that object. Meke kiynne assume B kiyla object ekk t1 transaction eken lock kra kiyl anit transaction vlt ba ek unlock krnn. Ek unlock krnn puluvn t1 transaction ekt vitri

Strict 2PL allows only serializable schedules

Strict 2PL does not allow. pahata pennl tiyenne strict 2pl use krl api kalin kata krmu problems 3 slove krn hati

WR conflict

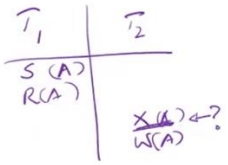
Meke kiyla tiyenne me vge seen ekk. Issellam t1 eke W(A) tiyenva. Rule 1 vlt anuva writing vlt kalin exclusive lock ekk tiyei. And t2 vala R(A) t kalin shared lock ekk tiyei. Eva X(A) and S(A) kiyl indicate krl tiyei. But mehi we can't use S(A) (shared lock) bcz it violates 3rd rule. (Its mean t1 A exclusive karat passe t2t ba ek S(A) krnn) So t2 is waiting t1 release exclusive lock. So t2 can't read A (R(A)).



To avoid Write Read conflict we can use use X lock before modifying then cannot Read that object before release lock

RW conflict

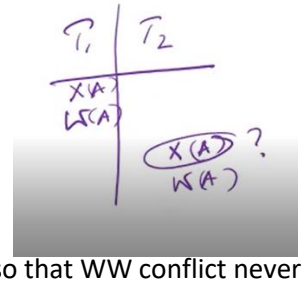
Meke venne arake anit patta. Ihata kata kra eke anit patta mehi vei. Its mean t2 can't X(A) bcz it also violates 3rd rule



This will also violate 3rd rule so that RW conflict never happen

WW conflict

Meket venne me vge deyk. T1 take X(A) and t2 ask for X(A). but it violates 3rd rule also



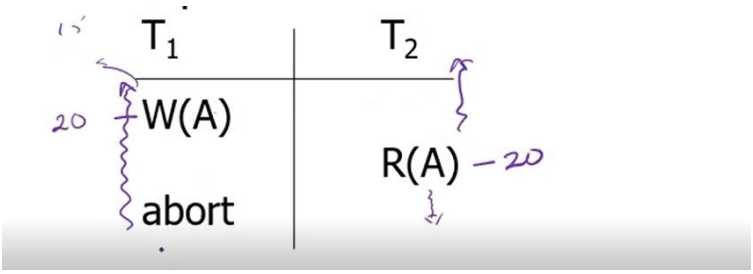
This will also violate 3rd rule so that WW conflict never happens

Aborting a Transaction

considerations with aborts

1. **Cascading aborts –** meke kiyl tiyenne me vge seen ekk. Assume initially A 15 and W(A) vlin ek 20k venva. so, R(A) also = 20 and after t1 is abort. Its means t1 roll back then A = 15 in t1. But in t2 A = 20. Bcz t2 is dependent on t1. Me vge scenario ekkdi t1 abort krddi t2 also abort krnva. Mekt tama cascading aborts kiynne. Meka hoda deyk nevei. So we have to avoid this. E kiynne mkkhri hetuvk nisa ek transaction ekk vens unot anit ekt vens krn ek varadi ne

    - If a transaction T1 is aborted, all its actions have to be undone.
    - Not only that, if T2 reads an object last written by T1, T2(depending on transaction also canceled) must be aborted as well! That we call cascading abort
    - This is not a good thing. database servers should avoid cascading aborts.



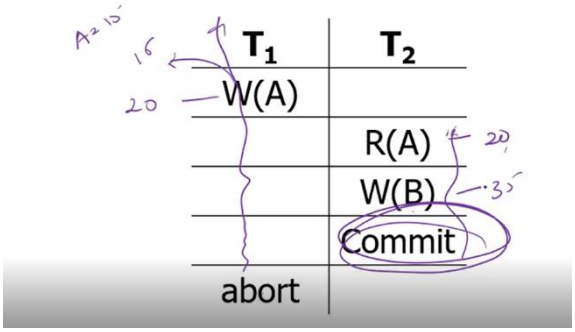
The root cause for cascading abort – (cascading abort happens venne hetuva WR conflict)

WR conflict

cascading aborts happen because of WR conflict

using strict 2PL protocol can avoid cascading aborts. The transaction which wrote the data aborts

2. Unrecoverable Schedules



Meke venne me vge seen ekak. A initially 15i. Assume t1 eke write ekedi A=20k venva. Then t2 A read karam 20 print venva. And W(B) = 35k venva. Then commit krnva. Its mean permanently save krnva. Then t1 abort krnva. Then A aaai 15k venva. But t2 vala A 20i. bcz ek commit krl tiyenne. Ek tama meke tiyena aula. Then apita t2 aaai recover krnn ba. Bcz ek commit krl tiyenne. Ek tama mekt unrecoverable schedule kiynne. Ek hriyata api ATM ekk salli gattam e transaction ek aaai cancel krnn ba vge seen ekak. Mekt strict 2pl protocol use krl solve krnn puluvn

The root cause for Unrecoverable Schedules

WR conflict
When depending transaction can't be canceled because of that transaction already committed. So we call that Unrecoverable Schedules.

using strict 2PL protocol can avoid Unrecoverable Schedules

Transactions and Concurrency Control – Part 2

Overheads of Lock-based CC Protocols

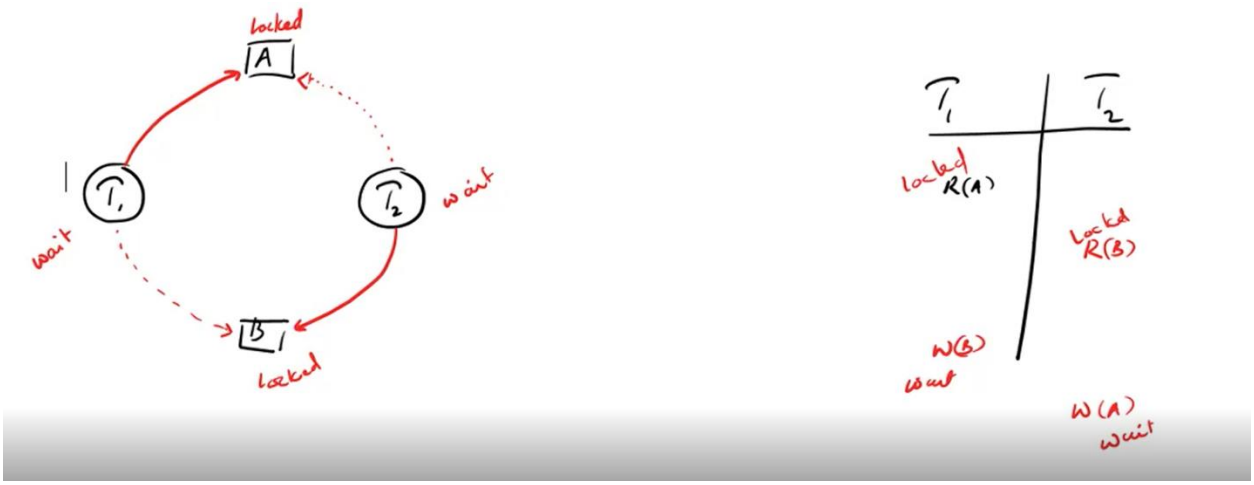
1. Handling deadlocks
2. Handling phantoms
3. Maintaining and handling locks

1. Handling Deadlock

The cycle of transactions waiting for locks to be released by each other. Ihata api kata kr strict 2PL protocol ek nisa tama deadlock ativenne

Saralavam qwot dead lock kiynne men mekt. Pahata rupayata anuva T1 eken A lock krl tiyenva and t2 eken B lock krl tiyenva. Then assume T1 t B access krnn oni. But ek krnn ba mkd T2 tama B lock krl tiyenne. So T1 witing venva T2 B release krnkm. E vgema T2 t oni A access krnn but ema krnn ba T1 A lock krl tiyenne. So T2 wait venva T1 A release krnkm. So meke tiyenne transaction 2i e dekam wait venva for each other. So meka nisa database ek struct venva. Mekt tama deadlock ek kiynne

When using Lock-based CC Protocols deadlock can happen.



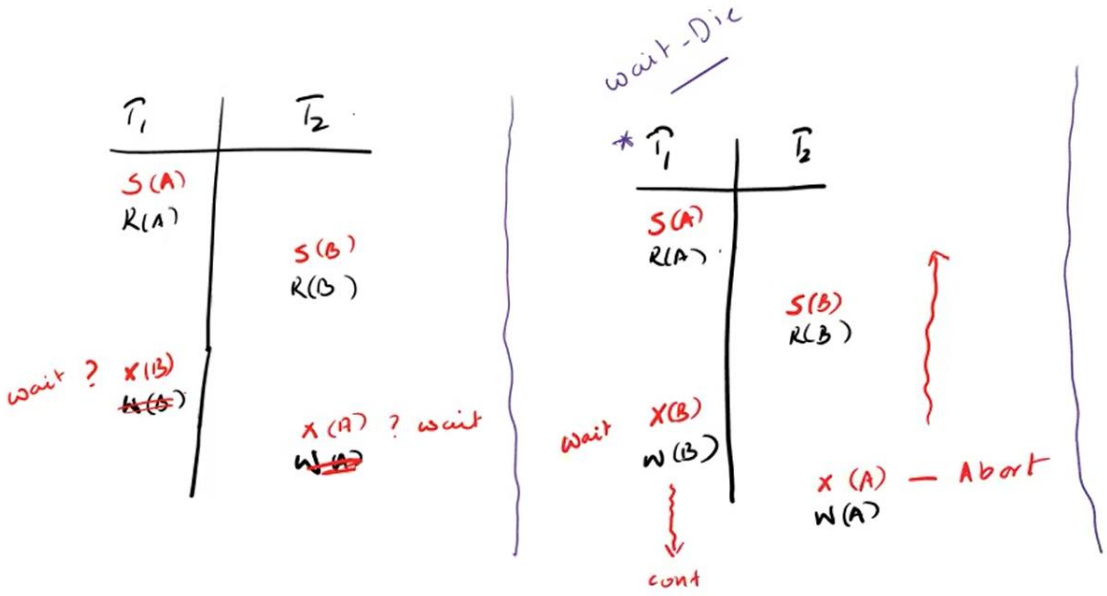
Deadlock ekk deal krnn akara 2k tiyenva.

Two ways of dealing with deadlocks

- **Deadlock prevention**
  Not allowed to happen deadlock
- **Deadlock detection**
  Identify deadlock after happened and remove them

1. Deadlock prevention

Problem eka: - meke T1 wait venva T2 B release krnkm. And also T2 wait venva T1 A release krnkm. Ehapatte pic eke pennala tiyenne me problem ek solve krnn wait die algorithm ek use krn hati



Wait-die (Deadlock prevention yatate tiyena Me algorithm ek use krl deadlock perevent krnn puluvn)

Wait die algorithm eke krnne transaction vala priority ek anuva evt timestpan ekk laba dena eka. Ehidi venne high priority transaction wait venva low priority transaction resource release krnkm. But if low priority transaction needs some resource that is already locked by high priority one the low priority one will it cancel.

Ihata rupeta anuva ek gattot meke T1 high priority one and T2 low priority one lesa gamu. Ehidi T1 ta X(B) krnn ba. Then T1 wait. T2 A ge **exclusive lcok ek release** puluvn. Bcz ek low priority ek. then ehidi venne T2 abort vena ek. Then t1 t process ek continue krn yann puluvn

Assign priorities based on timestamps
Assume Ti and Tj are transactions
2 policies are possible
Should assign priority for all the transactions.
Assign priorities based on timestamps (transaction start time).
The transaction is older one gives high priority.
The transaction is the newest one given low priority.

High priority transaction can wait till low priority release resources.

If low priority transaction needed resources locked by high priority transaction then low priority transaction will cancel.(low priority transactions cancel and released their resources).

First, we apply S2PL protocol

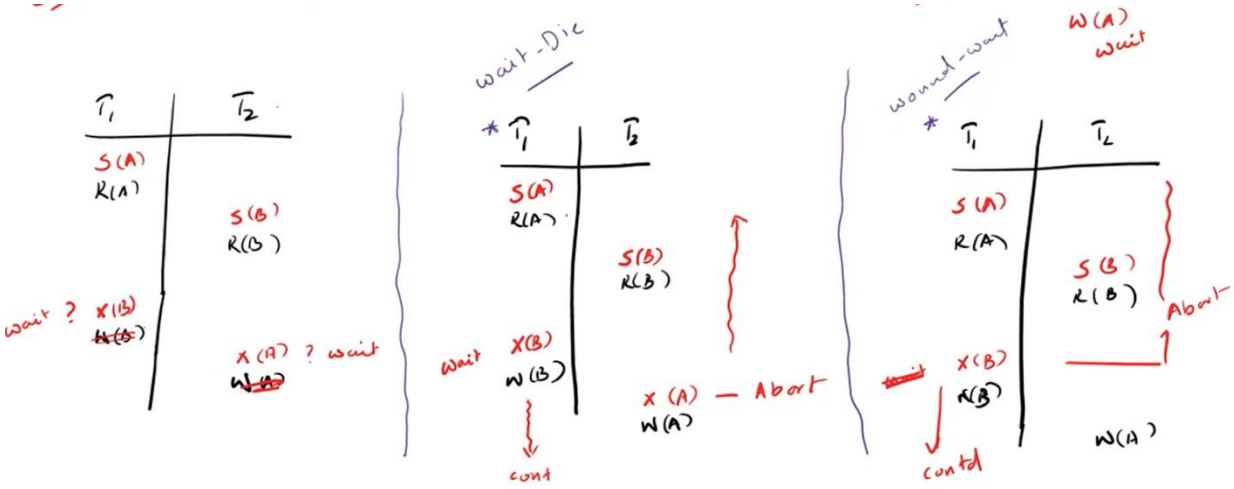
- Both of T1 and T2 transaction can't use X lock because it violated 3rd rule of S2PL protocol
- So X lock will wait
- Then happen deadlock situation
- To prevent deadlock, we apply Wait-Die policy

Apply Wait-Die policy

- According to the wait-die policy low priority can't wait for high priority locked resources released
- Then low priority transaction released and high-priority transaction can continue.

Wound wait (Me algorithm ekend deadlock prevent krnn puluvn. meke venne kalin eke anit patta. But saralava dekema venne low priority ek cancel vena eka. Its mean abort vena eka)

High priority transaction need resources locked by low priority transaction can cancel low priority transaction and get resources of low priority transaction and continue high priority transaction.



Me algorithm dekema saralavam venne high priority ek continue veddi low priority ek cancel vena ek. But low priority ekt api restart krnn oni. Mkd ekt userge mkkhri karyak krnnane tiyenne. Etkota venne meka. Hitann law priority ek 11 Am valadi cancel una. Then ek restart krnnet original timespans ekt. Its mean 11 AM. Then devani sare eka high priority ek venva

> If canceled Low priority transaction restart again should start at the original timestamps
> The reason for taking original timestamps is to get high priority for the canceled low-priority transaction.

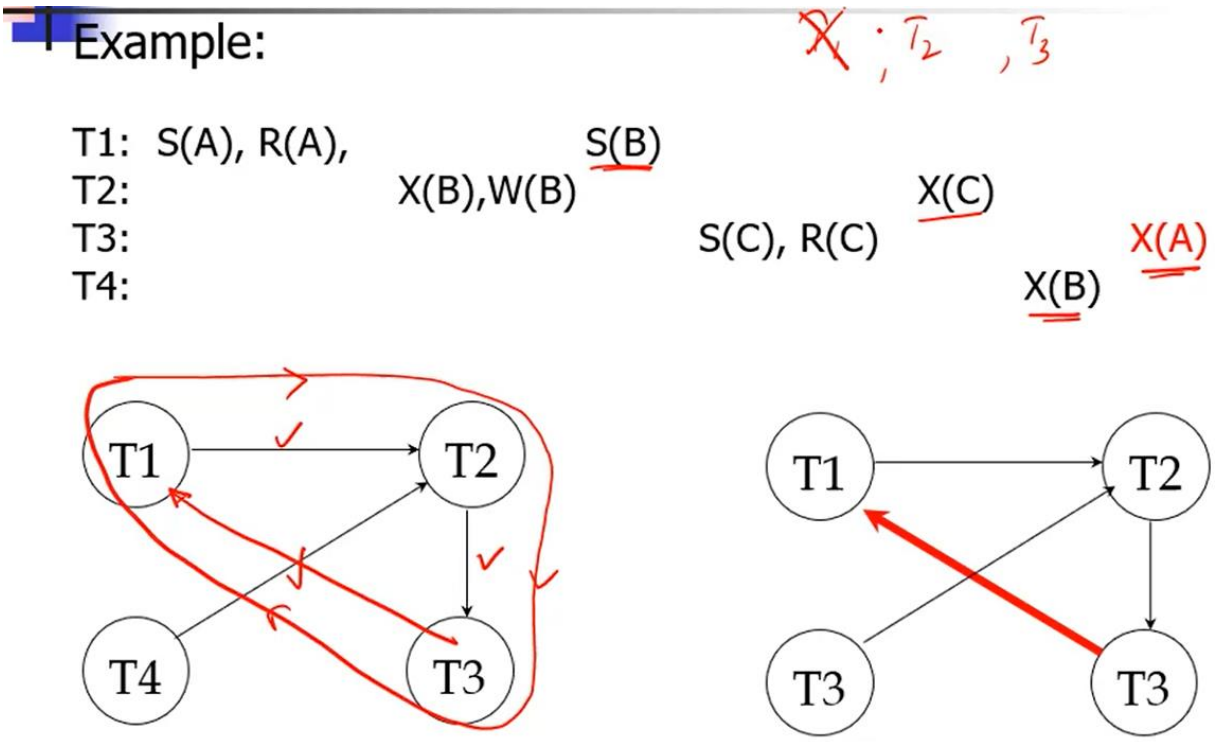
Ap me venk kata kre deadlock prevention gana dn kata krnne deadlock detection gana

2. Deadlock detection

Create a **waits-for graph** for identify deadlock

- Nodes are transactions
- There is an edge from T1 to T2 if T1 is waiting for T2 to release a lock
- We can identify deadlock after creating waits-for graph
- If deadlock occurs, pick a victim transaction and abort.

Saralavam me graph eke pennanne witing vena seen ekk. Ek transaction ekk tavat ekk venuven wait venvam ek me graph eke pennanava. Pahat rupe anuva T1 ge S(B) wait venva bcz T2 B t exclusive lcok ekk dala tiyenne. So ek tama T1 ge idl T2 adala tiyenne. Eke terume T1 is waiting for T2



2. Handling Phantoms

Dynamic Databases & Phantoms

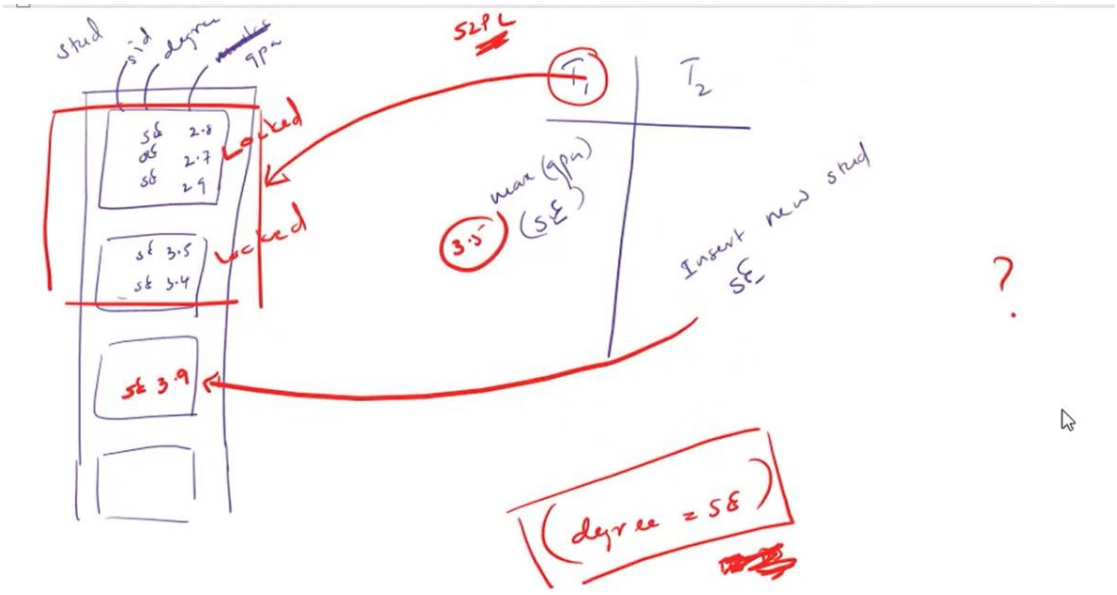
Saralavama static and dynamic kiynne hitanne db ekk record 1000l tiyei. Then apita eva retrieve krnn puluvn edit krnn puluvn ek static. But hitann DB ekt api new data ekk add krnva or tiyn ekk delete krnva eka dynamic. Api me venkm baluve strict 2PL protocol ek static databases vlt dana hati. But ek dynamic ekt set na. bcz dynamic databases vala records gana fixed na. eva growing and shirking venva

Strict 2PL is not enough for dynamic databases (because of growing and shrinking).
Strict 2pl can lock existing data in the database.

Phantoms

Saralavam qwot Phantoms problem kiynne mekt. Pahat pic eke tiyn ek gamu. Eke transaction dekak tiyei parellally run ven. T1 eken krnne SE lge highest GPA ek gann ek and T2 eken krnne alut SE kenek db ekt insert krn eka. Assume palaveni data blog deka t1 lock krl tiyenne eken highest GPA ek vidiyt 3.5 enva. But T2 eken alut SE kenek dnva palleha blog ekt eyge GPA ek 3.9. but highest ek vidiyt dila tiyenne 3.5 so there is a conflict. So, this may create an un-serializable schedule. This is all thing we call the Phantoms problem. Ekt hetuva tama In Strict 2pl lock can only lock existing data in the database. So, unlock area vlt anit transaction valin data insert krnn puluvn, but it conflicts with an ongoing transaction

> In Strict 2pl lock manager can only lock existing data in a database lock manager can't lock upcoming data in a database.
>
> Phantoms problem means unlocking areas other transactions can insert data, but it conflicts with ongoing other parallel transactions and creates an un-serializable schedule.



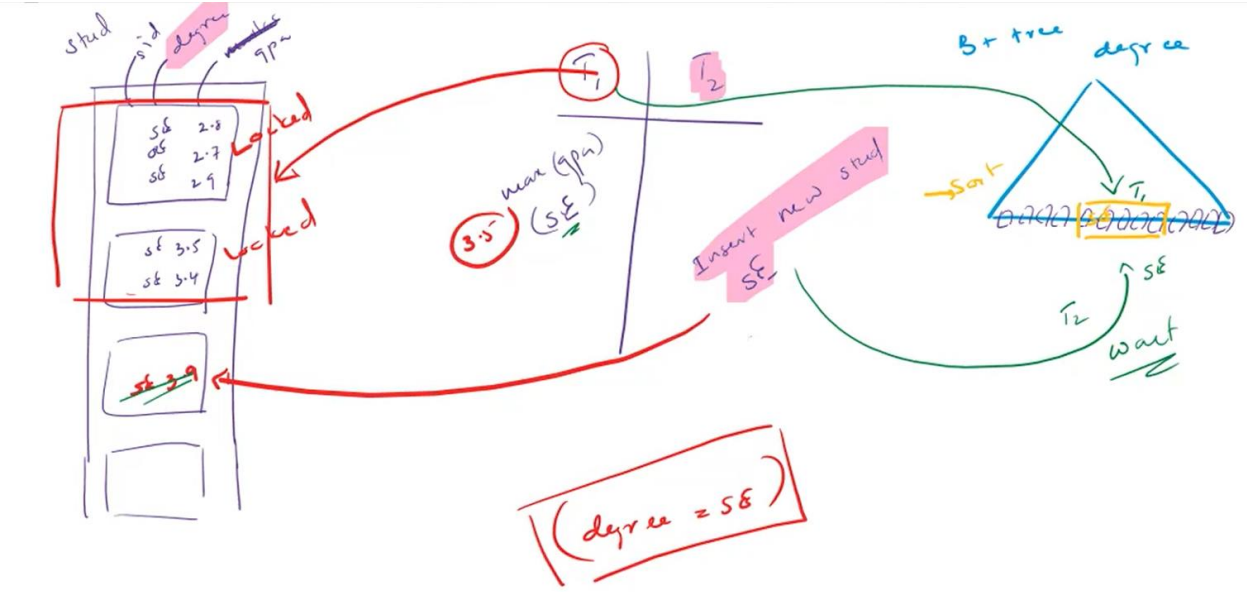
Thai conflict can be solved by various mechanisms.

1. **Predicate locking**
2. **A Simple Tree Locking Algorithm**

1. Predicate locking

Meka krnne memai. Attam insert krn student SE unot vitri ne conflict ek enne. DN or vena ekk unot aulk na ne. then api krnne then degree ek SE nm api insert krnn allow krnne na. nattan allow krnva. Me predicate ek use krot apita problem ek solve krgnn puluvn. Dn api blmu ek implement krn vidiya

Meka index locking use krl krnn puluvn. Ehdi krnne GPA column ek B+ tree index ekk hdn ek. Passe attam meke table eke vitrk nevei lock krnne B+ tree leaf level eke SE area ekt lock krnva. Then ehdid T2 transaction ek SE student kenek insert krnn yaddi ek T1 ek lock krl tiyenne. so T1 ek release krnkm T2 wait venva. Then above problem will solve

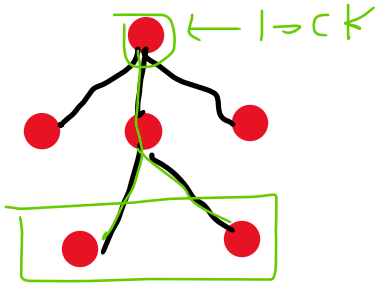


- Use **index locking** to implement predicate locking
- Create an index on a specific column (use B + tree)
- Index locking use to lock some leaf pages of B + tree
- When creating an index for a specific column that column values(entries) are in sorted order within the B+ tree leaf level
- Index locking lock all the related table values and b+ tree leaf level relevant values
- Using index locking can solve the Phantom's problem
- Index locking uses a tree structure (B+ tree)

Use S2PL

Then lock root levels and leaf levels

Api ihata kata seen eke tava aulk tiyei. Dn api B+ tree ek lock krnne strict 2pl protocol ek use krlnm eke tava aulk enva. Assume t1 transaction eken top nod ek lock kra kiyl. Then ehidi ek yatate tiyena okkoma node lock venva. Then anit ek transaction ekktvt anit node ekkvt use krnn bari venva.



For one transaction block all other transactions
So tree base structures S2PL Are not going to work

So mekt visadumak vidiyt tama simple tree locking algorithm ek enne

2. Simple tree-locking algorithm

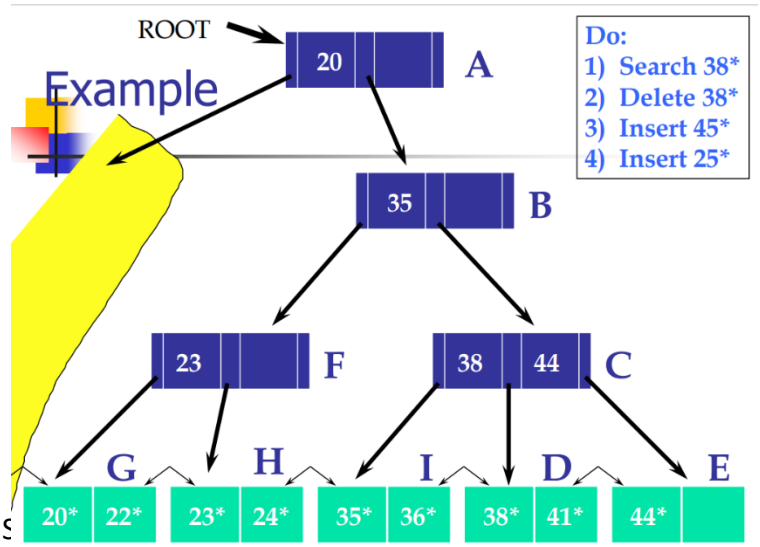
Meke search ek venne memai. After going to the child, release the parent. Child t giyama parent unlock krnva

Insert/ delete venne memai, Kalin eke vgem root eken patan gena pallehatta enva. Ehdid exclusive lock ek dagen enne. Bcz write ekk tiyen nisa. Root A nm X(A) then going to B then X(B). kalin eke vgema mehidida X(A) release krnn oni. But mehidi release krnn kalin api B ge child safe or notd blnn oni. Ehdi ek safe nm ita uda okkoma exclusive lock release venva. Dn blmu komada child safe or not d kiyl balnne komada kiyl. B+ matake gattam eke node ek full natnn apita ekt data insert krnn puluivn nattan alut node ekk hadagann oni. Then insert ekedi node ek full nattan api kiynva child is safe kiyla. Delete ekedi node ek half empty ekk nevei nm apita aulk natuva delete krnn puluvn ne. then delete ekedi node ek not half empty nm we say the child is safe. Ehidi ekt adala parents l okkoma release krnva

- Search – after going to child unlock parent (release parent)
- Insert/delete – before release parent should check the child is safe.
  If is child safe can release all the parent nodes(ancestors)
  How does the check child is safe or not
  Inserts: Node is not full.
  Deletes: Node is not half-empty

Mekt hoda example ekk pahata tiyei

Examples



Search 38

S(A)
S(B) release S(A)
S(C) release S(B)
S(D) release S(C)

Delete 38

X(A)
X(B) is not safe (Node is half-empty) So, me step ekedi parent release krnn ba. Its mean X(A)
X(C) safe (Node is not half-empty) release X(A) and X(B)
X(D) safe, release X(C)

Insert 45

X(A)
X(B) safe, release X(A)
X(C) not safe
X(E) safe, release X(C) and X(B)

Inset 25

X(A)
X(B)safe release X(A)
X(F)safe release X(B)
X(H)not safe
This modification can't do.

In simply, what we are checking as safe or not safe, if some modification happens in the bottom of the B+ tree or somewhere how many levels come up. So meken api blnne modification ek krnn puluvnda barida kiyn ekai

3. Maintain and handle lock

Api mechchr vela lata kara lock krnva release krnva kiya kiya. Attama meva manage krnne lock manager. He maintains the table called the lock table. Etkota meke tiyena aul tama palleha mention krl tiyenne. E kiynne mema lock managd krnna its mean lock ek release krnvd kiyl decide krnn tika velvk yana eka

How to manage locks in an efficient way
The lock manager maintains the lock table
Lock management use to decide lock can be released or not in quickly

Database ekk tiyenne hierarchy ekkt
Database-> tables-> pages-> tuples

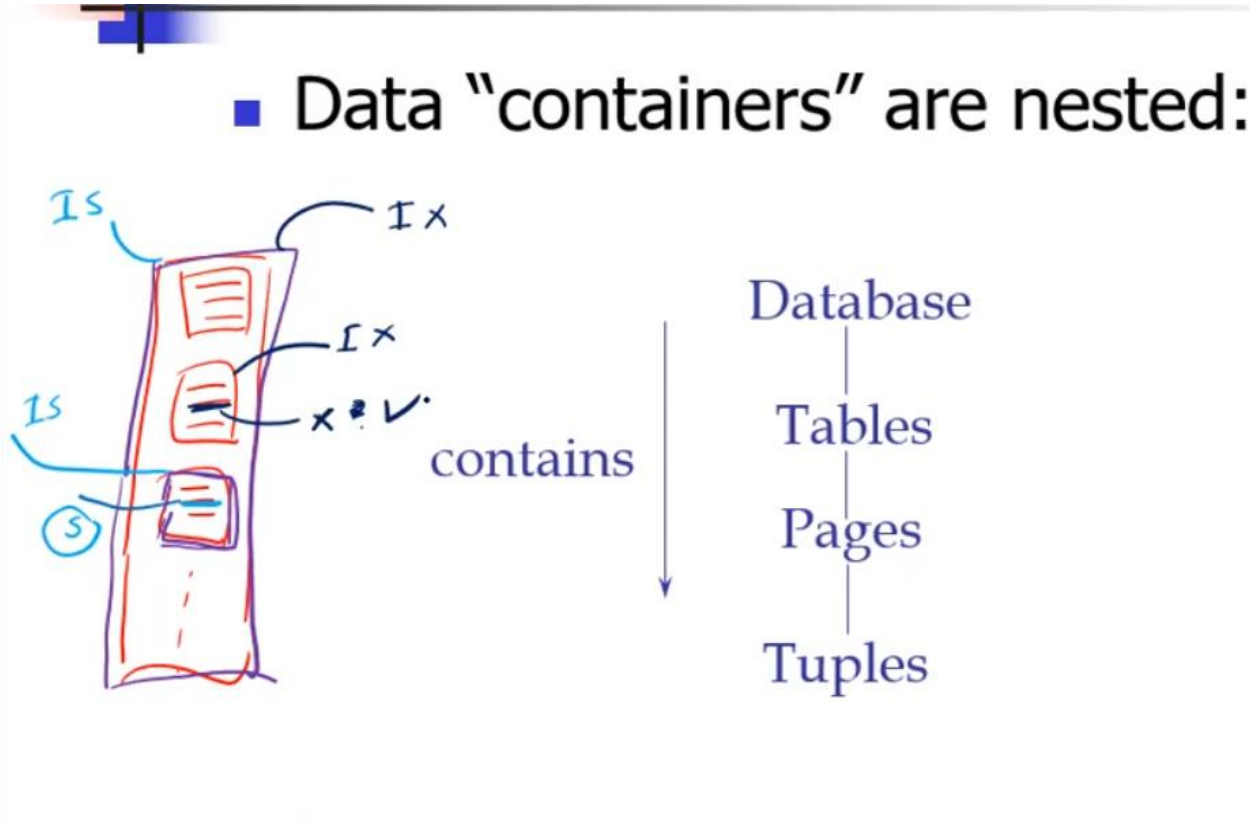
Multiple-Granularity Locking schema (meka use krl tama lock manage krnne)

Have 4 locks

IS-intention shared lock
IX – intention exclusive lock
S – shared lock
X- exclusive lock

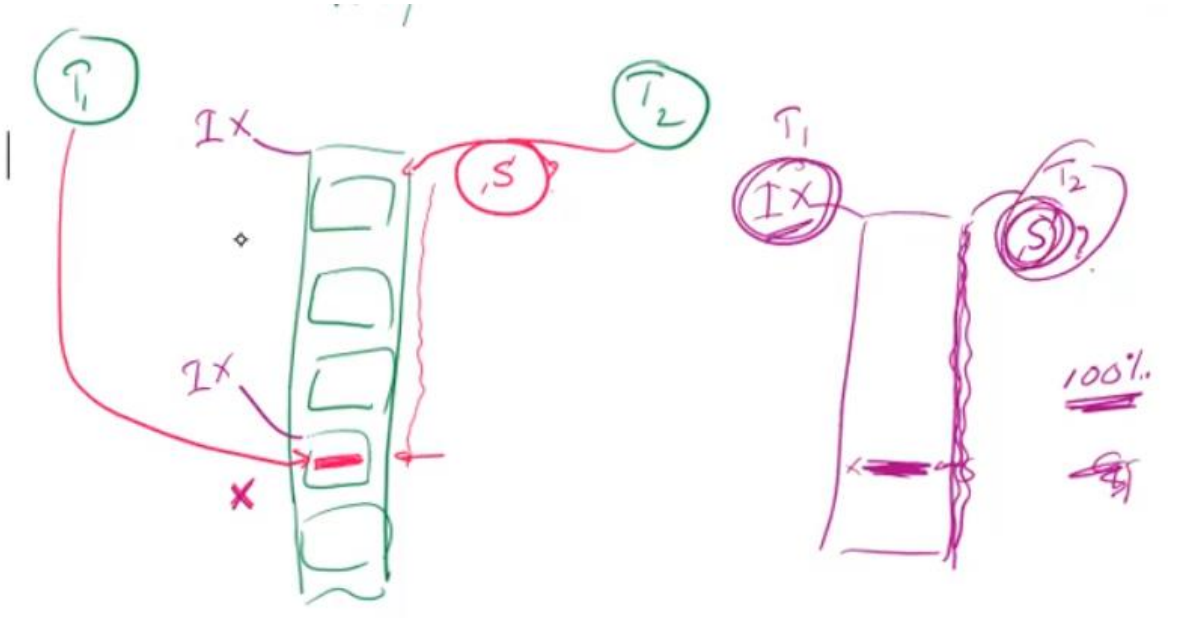
Meke behavior ek blmu. Pahat rupeta anuva,
assume we want to read some record. So ekt shared lock ek oni ne. before taking it we want to have an intention shared lock for all the higher levels. Higher level kiynne dn api inne tuple ekk eke higher level ek page ek eke table ek and after database ek. After finally we take shared lock for read it

E vidiytm apita mona hri modify krnnt oni nm. For that, we want an exclusive lock. Before taking it we have to take intention exclusive for page, table, and database. Finally, we can take exclusive lock for tuple



Dn api blmu ihata kata kl idea ek use krl komada efficiently locks manage krnne kiyl

Pahata rupeta anuva, T1 use krl tiyenne table akt data ekk write krnn. T2 use krl tiyenne table eke data okkoma read krnn. T1 wirte krddi eke table ekt intension exclusive lock ek vatenva. Ehidi T2 read krnn yaddi lock manager blnva same content ekt dila tiyn anit lock monada kiyla. Ehidi lock manager aduragannv IX dila tiyenva kiyla. Then lock manager dannav table eke kohe hari tanaka modification ekk venva. Exactly kohed kiyla danne nati unata kohe hari venva kiyla dannava. Then lock manager blnva T2 request krl tiyenne moketda kiyla. Ohu req krl tiyenne read all the data vlt. Then the lock manager identifies krgnnv anivaren T2 allow krot conflict ekk enva kiyla. So, lock manager IX release venkm T2 t data read krnn denne na. Then mehem ikmnt lcok manager decision gann puluvn. then ape problem ek solve venva



Lock manager access grant krnne or natte men me table ekt anuva. Access laba dena evt hriya hariya dala tiyenne access denne nati ev hisva tiyl tiyenne

intension

T2

T1

	--	IS	IX	S	X
--	√	√	√	√	√
IS	√	√	√	√	
IX	√	√	√		
S	√	√		√	
X	√				

Assume meke IS and IX access laba dila tiyenva. Ekt hetuva ek 100% na conflict ekk enva kiyl. Bcz eke table eke kohe hri kiyn idea ek denne. S X mean okkoma. S mean okkoma read krnn. Then IX and S access dila natte IS mean kohe hari data add venva S mean okkoma read krnva. Then 100% sure conflict ekk enva. IS and IX men kohe hri add krnva and kohe hari read krnva. So, 100% sure na conflict ekk ei kiyk.

SIX:- kiyl ekkut tiyei. Ek special ekk. Eke teruma share intension exclusive lock kiyna ek. Meka use krnne reading krn atartura changes krddi. EX: - update eke vge. Data read krl conf=dition ek satisfied nm update krnne. Ann ee vge seen ekt mek use krnn

- Isolation Level:
  - SERIALIZABLE (Strict 2PL + Index Locking)
  - REPEATABLE READ (Strict 2PL)
    - Phantoms possible
  - READ COMMITTED (S-locks are released b/f end)
    - Phantoms+ Unrepeatable reads possible
  - READ UNCOMMITTED (can only be used with READ ONLY transactions)
    - Phantoms + Unrepeatable reads + Dirty reads possible

Summary

- Overheads of lock-based CC protocols
  - Deadlocks
    - Prevention and Detection
  - Phantoms
    - Predicate and Index locking
  - Handling locks
    - Multiple Granularity Locking
- Controlling Isolation level using SQL