Database - Collection of related data

A database represents some aspect of real world

DBMS - Database Management System

Software to create, maintain and access data

Physical Level (Internal):
Physical Data model, data storage, access path

Logical Level (conceptual):
Describes entire database: entities, data types, relationships, ~~code~~ constraints
Physical & data independence

View Level (External):
describes the part of the database that a particular user group is interested in and hides rest.

~~Data isolation - Data separated from queries~~

Database Architectures

1) Centralized - Located and stored in a single ~~list~~ loc.

2) Client Server - Users connect to the database through network

3) Parallel - Multiple processors work in parallel on the database to provide resources.

4) Distributed :- Collection of multiple interconnected databases, which are spread physically across various locations that communicate via computer network.

Domain types in SQL

1) Char(n) : Fixed length character string
2) varchar(n)
3) int
4) smallint
5) Numeric (p,d) - Decimal size ~~at round~~ precision
6) real, double precision
7) float(n)
8) date : "yyyy-mm-dd" ex: 2005-7-27
9) ~~time~~ time
10) time stamp
11) Interval
12) blob - binary large object (photos, videos etc)
13) clob - char large object

ex: create table instructor (ID char(5), name varchar(20) not null, dept_name varchar(20), salary numeric(8,2))
↳ digits after decimal

- Superkey is an attribute that will identify any tuple must be unique

Relational Operators Algebra

- Selection - selecting rows - $\sigma_{B=17}(A)$ → match rows where B column as 17
- Projection - selecting columns - Π $\Pi_{A,C}(A)$ Duplicate rows removed
- Joining - Two or more tables colums
  ↳ Cartesian product - it forms all possible combinations → T1 rows x T2 rows = Total rows
- Union - combines 2 same columns from 2 different tables and only retains unique rows.
- Difference (R-S) → Tuple should be in R but not in S
- Intersection → Tuples common in 2 tables ($\cap$)
- Rename (ρ) - Rename a column while performing Cartesian product
- Natural Join - It joins tuple where the common column value matches
  i) Left outer join - Keep all from left and matching from right
  ii) Right outer join - Keep from right
  iii) Outer join - Only Common colums.

SQL

- entity integrity - ~~not null~~, unique
- foreign key references other tables

Ex: create table instructor (
    ID char(5),
    name varchar(20) not null,
    dept_name varchar(20),
    salary numeric(8,2),
    primary key (ID),
    foreign key (dept_name) references dept_name)
    table);

* reference ~~key~~ should be created first

* primary key declaration ensures attributes not null.

Ex: drop table instructor - deletes instructor table ~~only rows not the~~

- delete table instructor - Only deletes rows table structure remains intact.

- alter table instructor (add grade varchar(2))
to add columns.
to ~~do~~ drop use drop in alter table instruction.

SQL Query - select $A_1, A_2, \ldots A_n$
from $r_1, r_2, \ldots r_m$
where P.

String Operations

Like operator is used for pattern checking
i) % - It matches any substring
ii) Underscore (_) - It matches character

eg: Intro% looks for strings beginning with intro
___ looks for 3 char string

'100\%' used to compare substring including %

Concatenation ("||")

Pattern matching is case sensitive

eg: Select SUBSTR('ABCDEF', 2, 3) → BCD
↳ 3 chars from 2

Select * from instructor, teacher
↳ cartesian product

Select ID, name, salary/12 as monthly salary

Compare multiple columns
teacher.id
where (instructor.ID, dept_name) = ('CSE', 'CS')

- Select course_id from section where sem = 'Fall' and year = 2009.

AND → Union operation

Null Values

- The result of any arithmetic operation on null is null
  5 + NULL = NULL

Degree of Relationship set

- Binary relationships - involves two entity sets
- Relationships between more than 2 entity sets are rare ~~Most~~ They are called ternary

Attributes

- An entity is represented by a set of attributes
- Domain - The set of permitted values of each attribute
- Attribute types: Having more than one number
  i) Single & Composite
  ii) Single-valued and multi-valued (Phone nos)
  iii) Derived attributes - Can be computed from other attributes
  ~~Eg~~: Age from DOB

Mapping Cardinality Constraints

- Expresses the number of entities to which another entity can be associated via a relationship set
- Types of mapping cardinality for binary relationship
  i) one to one
  ii) One to many
  iii) Many to one
  iv) Many to many

- CHEN's notation to be used

- Rectangles – Entity sets
- Diamonds – Relationship sets
- Underline – Primary key
- Attributes listed inside entity rectangle – Ovals

→ One-cardinality
— Many-cardinality

- Discriminator or partial key underlined with a dashed line

- Entity is a key which has different attributes associated with it.

* If object can be expressed by its properties then its an entity.

- * A relationship is an association among several entities
  ex: 4453 (Austin) advisor 22222 (Einstein)
  student

Attribute Types:

- Simple and composite attributes
- Single-valued and multivalued attributes
- Derived attributes

Symbols

E → Entity set

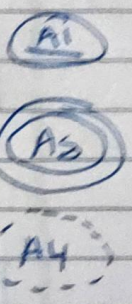
R → Relationship set

R → Identifying relationship set for weak entity set

R — E → Total participation of entity set in relationship.
↳ Total participation of values in E

E
A1
A2
A2.1
A2.2
{A3}
A4()

Attributes:
Simple (A1)
Composite (A2) and
Multivalued (A3)
Derived (A4)



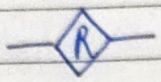
E
A1

Primary key

E
A1

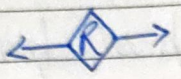
Discriminating attribute of weak entity set

↳ They don't have primary key and can't be identified on their own.

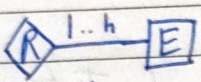


many-to-many relationship

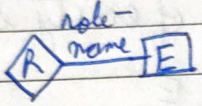
many-to-one relationship



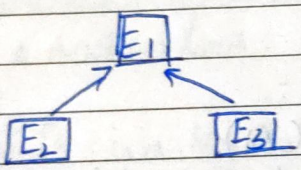
one-to-one relationship



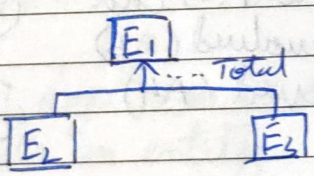
Cardinality limits



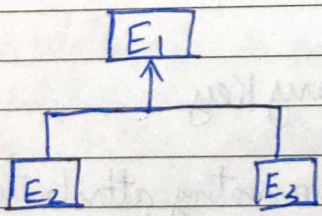
Role indicator



ISA: Generalization or specialization



total (disjoint) generalization



disjoint generalization

Intermediate SQL

NULL and Three Value

Any Comparison with NULL = NULL

Three Value Logic

OR : Unknown OR True = True
" Or False = Unknown
" or Unknown = Unknown

AND : Unknown and true = Unknown
" and false = false
" and Unknown = Unknown

NOT : (Not Unknown) = Unknown

Aggregate Functions

avg, min, max, sum, count

Using group by :

```
select dept_name, avg(salary) as avg_salary
from instructor
group by dept_name;
```

Gives aggregate for each dept_name

Having Clause with group by.

To get dept_name for having avg-salary > 4200

```
select dept_name, avg(salary) as avg_salary
from instructor
group by dept_name
having avg(salary) > 42000
```

☆ Having is used for statements having group by

☆ Where is used for general queries

Nested Subqueries

A subquery is a (select-from-where) expression that is nested within another query.

Set Membership:

Find courses offered in Fall 2017 and Spring 2018

```
select distinct course_id
from section
where semester = 'Fall' and year = 2017 and
course_id in (select course_id
              from section
              where semester = 'Spring' and year = 2018);
```

Not can be added to check opposite case

'Some' Clause

Ex: Find names of instructors with salary greater than that of some (at least one) instructor in Bio dept.

```
select name
from instructor
where salary > some (select salary
                     from instructor
                     where dept_name = 'Biology');
```

'All' Clause

Replace some with all salary value greater than others

Test for Empty Relations

- The exists construct returns the value true if the argument subquery is non empty

exists r ⇔ true if $r \neq \phi$
not exists r ⇔ true if $r = \phi$

- Except can be used to add exceptions to exists clause.
- Unique used to check if duplicate tuple is present in table.

'with' clause (*Not supported in many DBMS)

- It provides a way of defining a temporary relation whose definition is available only to the query in which the with clause occurs.

Scalar Subquery

- A subquery which returns only one value
- Runtime error if more than one tuple returned

Deletion

- Delete all from instructor : deletes all instructors
- Delete all instructor whose from Finance Department
  Delete from instructor where dept_name='Finance';

Insertion

Ex: Make each student in Music Department who earned more than 144 credit hours an instructor in the Music department with salary of $18000

```
Insert into Instructor
    Select ID, name, dept_name, 18000
    from Student
    where dept_name = 'Music' and total_creds > 144;
```

Update

Eg:1) Give a 5% salary hike to all instructors

```
Update instructor
set salary = salary * 1.05
```

ii) Give a 5% raise to those lesser than average

```
Update instructor
set salary = salary * 1.05
where (~~select salary~~ avg(salary)
salary < (select avg salary from instructor);
```

Case Statement

Eg:1) Hike 5% for less than 100000 and hike 3% ~~otherwise~~ ~~less~~ >100000

```
Update instructor
set salary = case
        when salary <=100000 then salary * 1.05
        else salary * 1.03
        end.
```

* Not equal <>

Natural or Conditional Joins

Natural join - Joins by matching column

Natural left outer join - Join by matching but keeps all tuples from left table

III for right outer join

Inner join → course inner join prereq on course.course_id = prereq.course_id.
→ Joined on condition column is repeated

Outer join retains left table tuples

Views

Temporary variables not stored such as tables created using select

eg: i) View of instructors without salary

create view faculty as select ID, name, dept_name from instructor

ii) To create custom columns

create view departments_total_salary (dept_name, total_salary) as select dept_name, sum(salary) from instructor group by dept_name;

View defined using other views

create view physics-fall_2009 as select course.course_id, sec_id, building, room_number from course, section where course.course-id = section.course-id and course.dept_name = 'Physics' and section.semester = 'Fall' and section.year = '2009';

To find courses only of watson

create view physics-fall-2009_watson as select course-id, room_number from physics-fall_2009 where building = 'Watson';

Insert statement is same when inserted in view it inserts in parent table also missing value takes up NULL value.

A view is created with values of history department insertion will fail if another subject is tried to be inserted. allow

To prevent the insertion check option should be used.

Materialized View - Table created with all the tuples present in parent table

Integrity constraints on single relation

- Not null

- Unique :

i) Unique specification allows the values to be form a candidate key.

ii) Candidate keys can be null (whereas primary key cant)

- Check (P)

Referential Integrity

- To show when value is updated in one table referenced table should be updated

- Create table "{

"

foreign key (dept_name) references department
on delete cascade
on update cascade,

)

Complex check clauses

create assertion <name> check (predicate);

Built in data types

- Date '2005-07-27'
- Time: '09:00:00.75'
- Timestamp: '2005-7-27 09:00:30.75'
- Interval: eg: Interval '1' day

User defined types:

eg: create type Rupee as numeric (12, 2)

Domain

- Data type with constraints

eg: create domain ~~construct~~ person-name char(20) not null.

Large-Object types:

- blob - binary large object
- clob - characters large object

Authorization Specification in SQL

- The grant statement is used to confer authorization

grant <privilege list> on
<relation name or view name> to <user list>

User list:

- a user id
- public
- A role

Ex: Grant select on instructor to U_1, U_2, U_3
" all on " to give all

- Revoking

- The revoke statement is used to revoke authorization
revoke <privilege list>
on <relation name or view name> from <user list>

Ex: ~~revoke selection on~~
revoke select on branch from U_1, U_2, U_3

Roles

create role instructor;

grant instructor to Amit;

↳ To add give someone the role

Roles given to other roles

ex: create role teaching_assistant

grant teaching_assistant to instructor

- Transfer of privileges

  grant select on department to Amit with grant option

  revoke select on department from Amit, Satoshi cascade

  ↳ If amit permission is revoked whoever he granted will also lose

  revoke select on department from Amit, Satoshi restrict

  ↳ Opposite of cascade

  grant reference (dept_name) on department to Mariano

  ↳ If dept_name is present in another table which he is editing he can also edit table in referenced table.

PL SQL

- Procedural Language extensions to SQL
- SQL is a popular language for both querying and updating data
- PL/SQL adds many procedural constructs to SQL language to overcome some limitations of SQL.

PL SQL Block

Declaration Section
Begin
Execution Section
Exception
Exception Section
END;

eg: DECLARE
I_Customer_group VARCHAR2(100) := 'Silver';
BEGIN
I_Customer_group := 'Gold';
DBMS_OUTPUT.PUT_LINE(I_Customer_group);
END;

ii) DECLARE
INAME Instructor.Name%Type;

INAME gets same data type as name in Instructor

```
Begin
  Select name from instructor where id = '12345';
  DBMS_output.put_line(name || ' ' || Salary);
END;
/
```

If then statement

```
DECLARE
  n_sales NUMBER:=2000000;
BEGIN
  IF n_sales>100000 THEN
    DBMS_OUTPUT.PUT_LINE('sales revenue
    is greater than 100k');
  END IF;
END;
```

Case Statement

```
... Declare ...
Begin
  c_grade:='B';
  CASE c_grade
  WHEN 'A' THEN
    c_rank:='Excellent'
  WHEN 'B' THEN
    c_rank:='Very Good';
  ELSE
    c_rank := 'No such grade';
  END CASE;
  DBMS_OUTPUT.PUT_LINE(c_rank);
END
```

```
LOOP
...

BEGIN
  LOOP
  I_Counter := I_counter + 1
  IF I_Counter > 3 THEN
     EXIT;
  END IF;
  dbms_output.put_line('...' || I_counter);
  END LOOP;
 -- Control resumes here after exit
END
```

```
FORLOOP
...
BEGIN
  FOR I_Counter in 1..5 or WHILE n_Counter <= 8
  LOOP
    DBMS_Output.Put_line(I_Counter);
  END LOOP)          (ncounter := n_counter + 1;
END;
```

EXIT when counter = 3
↳ Acts as break

CONTINUE same as usual

Procedure (~~function~~)

- It is a reusable unit that encapsulates specific business logic of that application
- It is a named block stored as schema object in Oracle database

eg:
```
CREATE [OR REPLACE] PROCEDURE procedure_name
(parameter_list) -> eg: dpt_name IN varchar
IS
[Declaration statements]
BEGIN
[Execution statements]
EXCEPTION
[Exception Handlers]
END [Procedure Name];
```

Parameters

i) IN - Read only, default, can be referenced but value can't be changed

ii) OUT - writable parameter, used for output

iii) INOUT - Readable and writable parameter

If not specified considered as IN

Function

- Almost same as procedure return statement is needed

```
CREATE OR REPLACE FUNCTION dept_count_fun(
dept_name IN varchar)
 RETURN number IS
  de_count number:=0
          L> declaration of data type
```

- Function must always return something
- Functions can be used in typical SQL statements like select, Update, Insert etc.

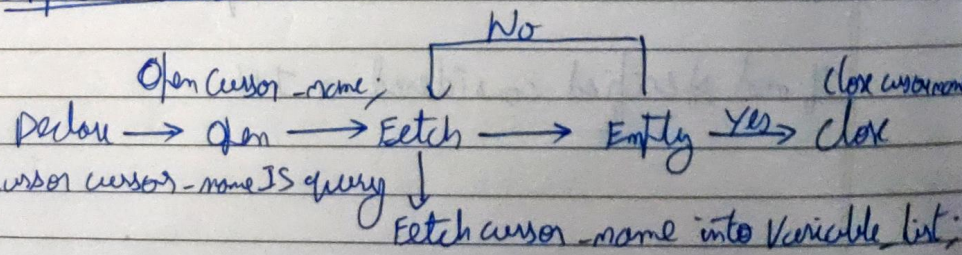
Cursor

- A pointer that points to a result of a query.

Implicit cursor:

When select into, insert, update and delete also automatically creates implicit cursors.

Explicit cursor:



↱ Refers to instructor table

```
Cursor c_instr IS select * From Instructor;
  n_instr c_instr%ROWTYPE;
begin
  OPEN c_instr;
LOOP
  FETCH c_instr INTO n_instr;
  EXIT WHEN c_instr%NOTFOUND;
   If n_instr.salary<60000 THEN
     Update instructor set salary= salary+10
     where id=instr.id
  ENDIF;
   DBMS_Output.Put_line ('If ...')
  ENDLOOP
  CLOSE c_instr
END;
```

↳ To be put inside a procedure or function

// Start of ER

papergrid
Date: / /

- Binary relationship - involve two entity sets
- An ~~of~~ entity is represented by a set of attributes
- Domain - set of permitted values for each attribute

Normalization

- Process to get schema into proper form
- Normal forms 1NF, 2NF, 3NF, BCNF

Functional Dependencies

α and β

$$\alpha \rightarrow \beta$$

For any tuples t_1 & t_2

$t_1[\alpha] = t_2[\alpha]$ then
$t_1[\beta] = t_2[\beta]$

eg:

	α	β
t_1	1	4
t_2	1	5
t_3	3	7

$\therefore \alpha \rightarrow \beta$ does not hold
True

Key - Value of other attribute can be uniquely identified using a key

Superkey - Identify all attributes in relation

Candidate Key - Minimal can't be broken down
$K \to R$ for which K is a candidate key then
$K \to R$ and $\alpha \subset K$ then $\alpha \to RX$
α should not uniquely identify R

$\alpha \to \beta$ is trivial, $\beta \subseteq \alpha$

Eg: (ID, Name) $\to$ ID
$\alpha \to \beta$
$\beta \subset \alpha$

	EmpID	Name	D-ID	DName
t_0	1	JD	1	HR
t_1	2	Jane	2	Marketing
t_2	3	John	1	HR
t_3	4	~~Global~~ Goodal	3	Sales

Check if $F: \{ EmpID \to Name \}$ Holds true
$\alpha \to \beta$

As they are all unique it holds true

$F: \{ D\_ID \to DName \}$ $t_1[1] = HR = t_2[1]$
$\alpha \to \beta$ $t_1[1] = HR$
$t_2[1] = HR$

$\therefore$ Holds true

☆ Closure of a set of functional dependencies (F^+)

$A \to B$ and $B \to C$ then $A \to C$
fd_1 fd_2 infers $fd_3 \to F^+$

Armstrong's Rules	Additional
R1 if $\beta \subseteq \alpha$ then $\alpha \to \beta$ (reflexivity)	R4: $\alpha \to \beta$ and $\alpha \to \gamma$ then $\alpha \to \beta\gamma$ (union)
R2 if $\alpha \to \beta$ then $\gamma\alpha \to \gamma\beta$ (augmentation)	R5: $\alpha \to \beta\gamma$ then $\alpha \to \beta$ and $\alpha \to \gamma$ (decomposition)
R3 if $\alpha \to \beta$ and $\beta \to \gamma$ we have $\alpha \to \gamma$ (Transitivity)	and R6: $\alpha \to \beta$ $\gamma\beta \to \delta$ then $\gamma\alpha \to \delta$ (Pseudotransitivity)

Eg: $R = (A, B, C, D, E, F, G, H, I)$

F: { $AB \to E$, $AG \to I$, $BE \to I$, $E \to G$
$GI \to H$

Can $AB \to GH$ be infered?

$AB \to E$, ~~$E \to G \Rightarrow AB \to G$~~

$ABB \to BE = AB \to BE$ — (5)

$AB \to BE \to I \Rightarrow AB \to I$. — (6)

$AB \rightarrow E$ and $E \rightarrow G \Rightarrow AB \rightarrow G$ — (7)

From (6) and (7)

$AB \rightarrow IG$ (union) — (8)

we have $GI \rightarrow H \Rightarrow AB \rightarrow H$ — (9)

From (7) and (9)

By union $AB \rightarrow GH$

$F^+ = \{F, 6, 7, 8, 9, 10\}$

2) $F = \{ A \rightarrow BC,\ CD \rightarrow E,\ E \rightarrow C,\ D \rightarrow AEH,\ ABH \rightarrow BD,\ DH \rightarrow BC \}$ or $R = \{A, B, C, D, E, H\}$

Show ~~DEH~~ $BCD \rightarrow H$ hold

$CD \rightarrow E$, $D \rightarrow AEH$, $CD \rightarrow CAEH$

$BCD \Rightarrow BCAEH$

By decomposition

$BCD \rightarrow H$ (Decomposition)

3) $A \to BC$
$B \to E$
$CD \to EF$
ST $AD \to F$

~~$CD \to EF \Rightarrow D \to$~~

$AD \to BCD$
~~$BCD \to BEF$~~
$AD \to BEF$

$A \to BC \Rightarrow A \to B$ and $A \to C$ — (4)

$AD \to CD$ — (5)

~~From~~ $CD \to EF \Rightarrow CD \to F$ — (6)

From (5) & (6)

~~Et~~ $AD \to F$

$F+ = \{$
$A \to BC$
$B \to E$
$CD \to EF$
$A \to B$
$A \to C$
$AD \to CD$
~~$A \to C$~~ $CD \to E$
$CD \to F$
$AD \to F$
$\}$

Attribute Closures

α under F

α^+ set of all attributes that can determine F

If $\alpha \to \beta$; $\beta \subseteq \alpha$

Ex: R(A, B, C, G, H, I)

$A \to B$, $A \to C$, $CG \to H$, $CG \to I$, $B \to H$

Find $(AG)^+ = \{AG$ (B) C (H) (I) $\to$ (G $\to$ I

a) AG $\to$ AG

↳ using (G $\to$ H

↓

A $\to$ B, A $\to$ C

$\Rightarrow$ Final $(AG)^+ = AGBCHI$

Uses

1) Check if FDs hold
2) ~~Find~~ candidate key and Superkeys
   PK
3) prime attributes and non prime attributes

Eg1: $R(A,B,C,D,E,F,G)$

fds: $AB \to CD$, $AF \to D$, $DE \to F$, $C \to G$, $F \to E$, $G \to A$

Does $CF \to DF$ hold?

CF^+ = CEGEAD

$\therefore CF \to DF$ is true

Eg 2) Find the candidate keys and super keys
$R(A,B,C,D,E)$

fds: $AB \to C$, $DE \to B$, $CD \to E$

By seeing FDs as AD can be seen as only on left side so start with them

$A^+ = A$
$D^+ = D$
$AD^+ = AD$
$ADB = ADBC \to AB \to C$
↳ ABCE | ADBCE
$ADC^+ = ADCEB$
$ADE^+ = ADEBC$

Note if $\alpha \to R$; α is a candidate key

In this case ADB^+, ADC^+, ADE^+ as they define all. Any one of these candidate keys can be Primary key.

Superkeys :- Superset of candidate key

ADB, ADBC, ADBE, ADBCE
ADC, ADCE, . . .

similarly all can be formed.

Prime attributes: PA = { A, B, C, D, E }

Non prime = { }

AD all are prime

Q2: R(A, B, C, D, E, F)
AB → C, CF → D, D → E, F → B, E → F

Find candidate, superkeys, prime attributes non prime

A+ = A → If this is candidate key don't add more letters
AB+ = ABCDEF
ABC+ = ABCDEF
ABD+ = ABDCEF
ABE+ = ABEFCD → If any letter missing that is non prime
AC = ACDEFB
AD = ADEFBC
AE = AEFBCD
AF = AFBCDE
CK = AB, AC, AD, AE, AF
Superkeys = AB, ABC, ABD, ABE, ABF, ABCD, ABCE . . .

Prime Attributes = PA = {A, B, C, D, E, F}

Non Prime = { }

Canonical Cover

Eg1: $A \to B$ $B \to C$ $A \to C$ ← Redendant fd

Eg2: $A \to B, B \to C, A \to CD$

↳ $A \to C$ Redendant

$A \to D$ → More essential

Extraneous Attribute

$\alpha \to \beta$ in F

i) A as extraneous in α and F logically implies
$F - (\alpha \to B) \cup ((\alpha - A) \to B)$

~~ii)~~

ii) A as extraneous in β

$(F - (\alpha \to \beta)) \cup \alpha \to (\beta - A)$ logically implyF

'A' Test for extraneous attribute

1) If $A \in \alpha$

i) Complete $(\alpha - A)^+$ using F

ii) Check if $(\alpha - A)^+$ contain β, then A is extraneous

LHS

2) If $A \in \beta$

i) Compute α^+ using $F' = F - (\alpha \to \beta) \cup \alpha \to (\beta - A)$

ii) Check α^+ contains A, if so A is extraneous

RHS

Eg 1) R(A, B, C)

$A \to BC$ — (1)

$B \to C$ — (2)

$AB \to C$ (3) $A \to B$ (4)

Find the canonical cover

i) Redundant fd's or parts

ii) Extraneous attribute removed

iii) Step I is repeated

A.) In (1) & (4) we observe (4) can be inferred from (1) so 4 is redundant

Now set

$A \to BC$ — (1)

$B \to C$ — (2)

$AB \to C$ — (3)

Checking ① & ② for extraneous

$\alpha \to \beta$

$AB \to C$

Check if A is extraneous

~~Check if B is extraneous~~

Check $(\alpha - A)^+ = B$

$B^+ = BC$

Even after removing A, BC is obtained

∴ A is extraneous.

Check if B is extraneous - no check needed as at least one attribute needed.

Now ② becomes $B \to C$ is available as ② hence should be removed.

New $fd = A \to BC$ — ①

~~$B \to C$~~ — ②

Check extraneous in attribute in ① for RHS

$A^+ = AC$ for $F' \Rightarrow A \to C$

↳ As B is not present here it is not extraneous

F'_2 for C $A \to B$

$B \to C$

$A^+ = ABC$ and C is available so C is extraneous.

Final canonical cover

$A \to B$

~~$B \to C$~~

Qn 2 R(A, B, C, D)
fd = { AB → CD , BC → D }

Find canonical cover

A.) Step 1: check for redundant fds:
No redundant fd.

Step 2: Check extraneous
If present remove

Equivalent or not

F and G are fd sets

F can be inferred using G and
if G can be inferred using F

then F = G

F covers G: If all fds of G can be inferred by F.

G covers F: If all fds of F can be inferred by G.

F covers G and G covers F then
F equivalent to G.

Eg1: F: { A → C — ①
AC → D — ②
E → AD — ③
E → H — ④

G: { A → CD
E → AH

If G covers F using Fds in G

A^+ = ACD → ① & ② of F is covered

E^+ = EAH → ④ of F is inferred

G covers F

If F covers G

Using (1) of F

$A \rightarrow AC$

Using (2) of F

$AC \rightarrow ACD \rightarrow$ (1) of G is inferred

Using (3) of F

$E \rightarrow EAD \rightarrow$ Using (4) of F

$EAD \rightarrow$ (2) of G is inferred

Lossless Join Decomposition

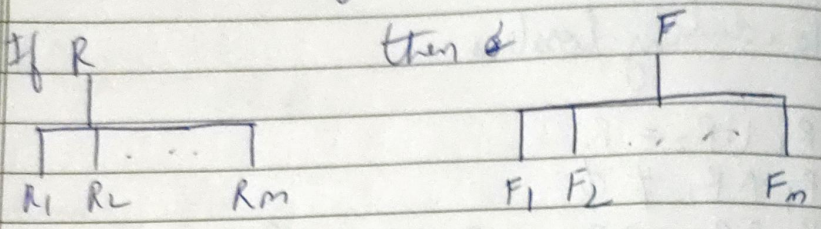
To identify lossless

- $R_1 \cup R_2 = R$
- $R_1 \cap R_2 \neq \phi$
- $R_1 \cap R_2 \rightarrow R_1$ or $R_1 \cap R_2 \rightarrow R_2$

Eg: $R = \{A, B, C\}$
$F = \{A \rightarrow B, B \rightarrow C\}$
can be decomposed in 2 different ways
$R_1 = (A, B)$, $R_2 = (B, C)$
Lossless join
$R_1 \cap R_2 = \{B\}$ and $B^+ = BC$

$R_1 = (A, B)$ $R_2 = (A, C)$
$R_1 \cap R_2 = \{A\}$ and $A^+ = AB$
Not dependency preserving
(cannot check $B \rightarrow C$ without computing $R_1 \bowtie R_2$)

Dependency Preserving



A decomposition is dependency preserving if
$(F_1 \cup F_2 \cup F_3 \ldots \cup F_m) \equiv F^+$

Common attribute should be used as superkey in order to preserve dependencies

To check dependency preserving:

1) ~~$F_1 \cup F_2 = F$~~

Q: R (A B C D E F)

F: A → BCD, A → EF, BC → AD, BC → E, ~~BC → F~~, B → F
D → E

→ New tables.

D = {ABCD, BF, DE}

F1	F2	F3
ABCD (R1)	BF (R2)	DE (R3)
A → BCD	B → F	D → E
BC → AD		

Need to check for A → BCD, ~~A → EF~~, BC → AD
~~BC → E~~, ~~BC → F~~, B → F, D → E
only one not in table need to be found

$(BC)^+ | F1 = (ABCD)^+ | F2 = (BFADE) | F3 = \underline{BCADFE}$

(A^+) $|F_1 = (ABCD)^+ | F_2 = (ABCPF)^+ | F_3 = ABCDPF$

∴ It is dependency preserving

Normal Forms

MPurpose:

- Eliminating redendent (useless) data
- Ensuring data dependencies make sense.

Anomalies

1) Update Anamoly - Deconpose the schema

Employee table having ID, address and skill.
If employee has ~~multiple~~ one address and ~~one skill~~ multiple skills then everytime ~~skill~~ is address is updated it might not reflect everywhere.

Better to deconpose into (ID, Address), (ID, Skill)

2) Insertion Anomaly value can't be entered into table due to null value or other constraint

3) Deletion Constraint if faculty has no courses assigned his value would get deleted.

1NF

- All relations only contain Atomic values
- Atomic values - Single valued attributes
- This form has lot of redundancies

2NF

- Checks:
  - → It should be 1NF
  - → Should not contain Partial Dependency

A value part of candidate key if it defines non prime attribute it is not in 2NF

3NF

- Should be in 2NF
- It should not contain transitive dependencies

$A \to B \quad B \to C \Rightarrow A \to C$ Should be avoided

Converting into 3NF

Given: relation R, set F of functional dependencies

Find: Decomposition of R into a set of 3NF relations R_i

Steps:

1) Eliminate redundant FDs, resulting in a canonical cover F_c of F

2) Create a relation $R_i = XY$ for each FD $X \to Y$ in F_c

3) If the key K of R doesn't occur in any relation R_i, create one more relation $R_i = K$

BCNF — Boyce Codd Normal Form

- Doesn't guarantee lossless join and dependency preserving
- Guarantees lossless join

Q1 R(N, D, P, S, A)

$N \rightarrow D$: fd1
$S \rightarrow A$: fd2
$(NS) \rightarrow P$: fd3

a) What is highest possible normal form/get relation to 2NF, 3NF, BCNF

1) Check for 2NF

Candidate Key = (N, S)

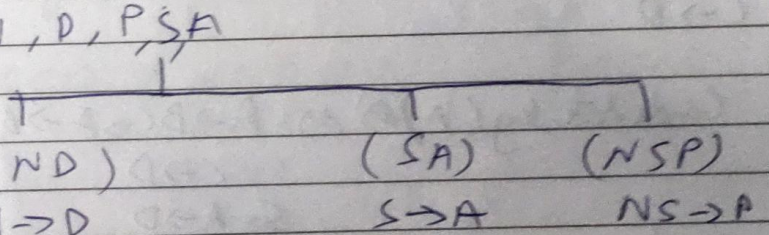
fd1 = Violation of 2NF as part of it identifies individually non prime attribute

$N \rightarrow D$
~~$S \rightarrow A$~~

fd2 = Not in 2NF

fd3 = In 2NF

∴ R is not in 2NF



is in 3NF
As all dependency satisfied so dependency preserving.

Qn2. R(W, X, Y, Z)

XY → Z – fd1

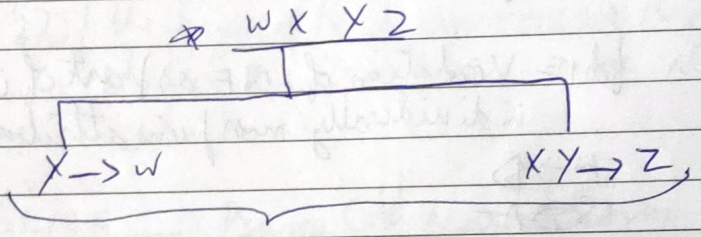
Y → W – fd2

candidate key (X, Y)

fd1 → Not in 2NF

fd2 → Not in 2NF as individually mapping to W

R not in 2NF



AD only 1 fd just to decide 3NF and dependency preserving

Qn3. R (A B C D)

A → BC

C → D

Candidate key (A) as A → BC ⇒ A → B and A → C

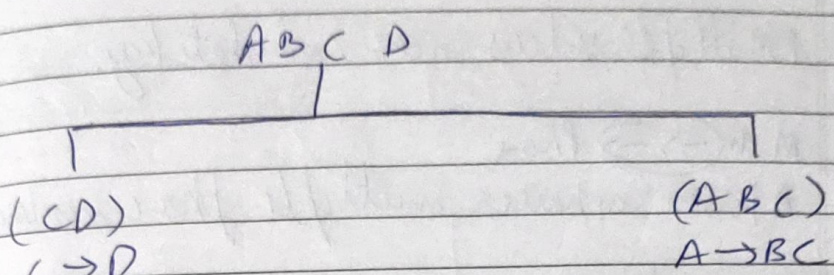
C → D

So A → D

Therefore as C → D is C is not candidate key

It is in 2NF

fd2 Violates 3NF as NPA → NPA



Q. R(CSJDPQV)

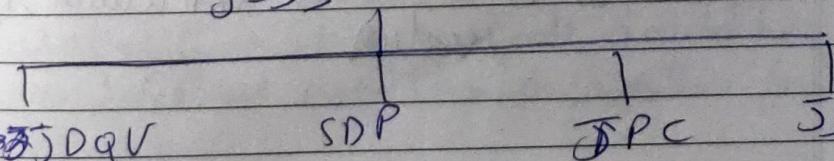
fd1 ⇒ C → SJDPQV
fd2 = SD → P
fd3 ⇒ JP → C
fd4 ⇒ J → S

Candidate key is C, So fd2, fd3, 4 are not in 3NF

AJ → S — (1)
~~SD → P~~ — (2)
JD → P can be obtained

For each fd in cc make a relation if key is not in any decomposed table then create new relation having key.

CC ⇒ C → DQV
SD → P
JP → C
J → S



Only one dependency in each case so BCNF

Multivalued Dependency

F Multiple values in one candidate key.

Man →→ Phone
A person can have multiple phone numbers.

Theory of MVDs

Complementation: If $X \to\to Y$ then $X \to\to \{R - (XUY)\}$

Augmentation: $X \to\to Y$ and $W \geq Z$ then $WX \to\to YZ$

Transitivity: $X \to\to Y$ and $Y \to\to Z$ then
$X \to\to (Z - Y)$

Replication: $X \to X$ then $X \to\to Y$ but reverse not true

Coalescence: $X \to\to Y$ and there is W such that $W \cap Y$ is empty, $W \to Z$ and $Y \geq Z$ then $X \to Z$

A MVD $X \to\to Y$ in R is trivial if

Y is a subset of X ($X \geq Y$) or

$X \cup Y$ = R otherwise non trivial repeating values

Transactions

A series of one or more SQL statements that are logically related are termed as Transaction

Eg: Transaction to transfer 50 ₹ for Account A to B

1) Read (A)
2) A = A - 50
3) Write (A)
4) Read (B)
5) B = B + 50
6) Write (B)

★ ACID properties

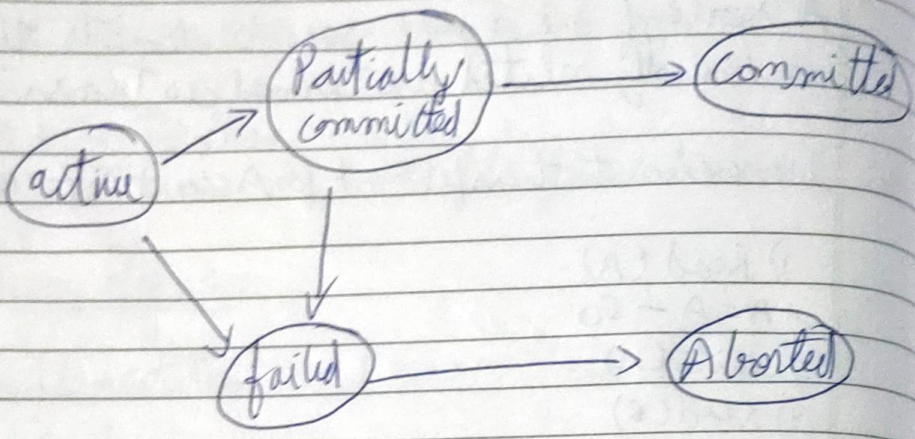
1) Atomicity - All operations should be completely executed or none

2) Consistency - Execution of a transaction in isolation preserves consistency of database
Eg: When 50 dollars transferred to B then B must be incremented

3) Isolation - Multiple executions taking place on a singular database. Transactions should be conducted in isolation

4) Durability - Changes made should persist in database even though system fails.

Transaction Model



Commit and Rollback commands

- A transaction begins with first executable SQL statement after a commit, rollback or connection made to a database.
- Transaction can be closed using commit or rollback.
- Commit ends current transaction and makes permanent changes
- Rollback ends transaction but undoes any changes made during transaction.

Syntax

```
Commit;
Rollback;
```

- Savepoint marks and saves the current point in the processing of a transaction.

savepoint deposit;
Rollback to savepoint deposit;

The Lost Update Problem

- This occurs when two transactions that access the same database have their operations interleaved in a way that makes the value of some database item incorrect.

The Temporary Update Problem (Dirty Read)

- This occurs when one transaction updates a database item and then transaction fails for some reason.

Incorrect Summary Problem

If one transaction is calculating an aggregate summary function on a number of records while other transactions are updating some of these records.

Schedules

Schedule - A sequence of instructions that specify the chronological order in which instructions of concurrent transactions are executed.

A schedule S of transaction $T_1, T_2, \ldots T_n$ is an ordering of all operations in these transaction.

Eg. Given
$T_1 = R_1(Q)\ W_1(Q)$ $\quad T_2 = R_2(Q)\ W_2(Q)$
(the order must be maintained)

- a schedule $R_1(Q)\ R_2(Q)\ W_1(Q)\ W_2(Q)$

Concurrent schedules used in real life situations.

- A concurrent schedule is serializable if it is equivalent to a serial schedule.

Serializability methods:

1) Conflict Serializable
2) Views Serializable

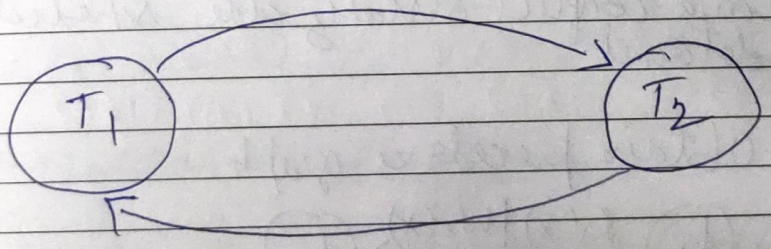
Conflict Serializability

1) ~~Read(Q)~~ t_i = Read(Q)

If any schedule S can be transformed into a schedule S' by a series of swaps of non conflicting instructions, we say that S and S' are conflict equivalent.

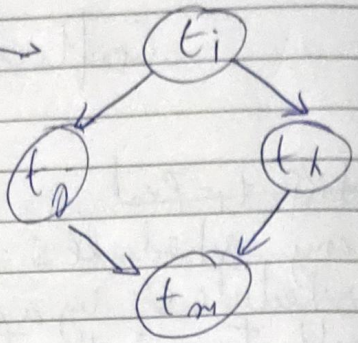
We say schedule S is conflict serializable if it is conflict equivalent to a serial schedule.

Precedence Graph - A directed graph where vertices are transactions



- A schedule is conflict serializable if and only if its precedence graph is acyclic
- If precedence graph is acyclic, the serializable order can be obtained by a topological sorting of the graph.

Topological sort →



1) Take node with no in coming edges & remove t_i
O/P => t_i

2) Repeat one
O/P: t_i t_j

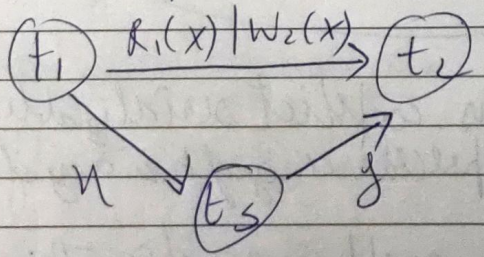
3) O/P t_i t_j t_k t_m

There can be more than one orders

Ex: $R_1(X)$ $R_2(Y)$ $R_3(Y)$ $W_2(Y)$ $W_1(X)$
$W_3(X)$ $R_2(X)$ $W_2(X)$

Can a conflict serializable schedule be obtained

Step 1: Obtain precedence graph



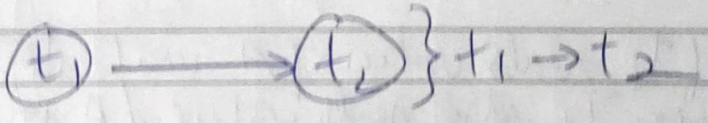
No loop, so no conflict

Note: Topological sort

$t_1 \to t_2 \to t_3$

Soln $R_1(X)\ W_1(X)\ R_3(Y)\ W_3(X)\ R_2(Y)\ W_2(X)\ R_2(X)\ W_2(X)$

Q. $R_1(A)\ W_1(A)\ R_2(A)\ W_2(A)$
$R_1(B)\ W_1(B)\ R_2(B)\ W_2(B)$



Sol" $R_1(A)\ W_1(A)\ R_1(B)\ W_1(B)\ R_2(A)\ W_2(A)$
$R_2(B)\ W_2(B)$

Ans: $W_1(X)\ W_2(X)\ W_2(Y)\ W_1(Y)\ W_3(Y)$

No precedence graph cyclic

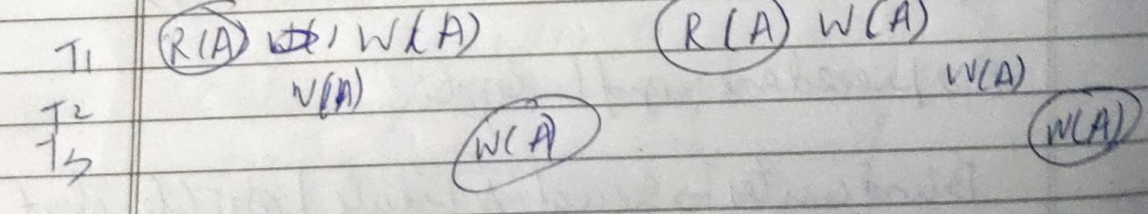
Blind writes → Serial Schedule possible about conflict serializable not possible.

When conflict serializable not possible try view serializable schedule

View Serializability

Schedules S_1 and S_2 are view equivalent if:

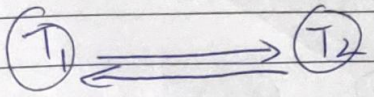
- T_i reads initial value of A in S_1, then T_i also reads initial value of A in S_2
- If T_i reads value of A written by T_j in S_1 then T_i also reads value of A written by T_j in S_2
- If T_i reads final value of A in S_1 then T_i also writes final value of A in S_2



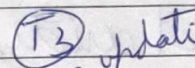
- Every conflict serializable is also view serializable
- Every view serializable that is not conflict serializable is having writes.

Qn: $R_1(A)\ W_2(A)\ W_1(A)\ W_3(A)$

If it is view serializable
If yes give its view serializable



Therefore not conflict serializable



→ Means no updation before

Step 1:

	A
initial read	T_1
Final Update	T_3
All writes	T_2 T_1 T_3

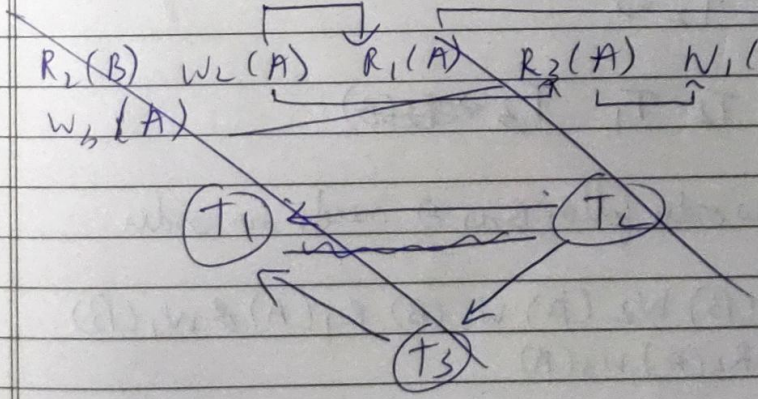
Read has to take place first

A: T_1 T_2 T_3 → Final write

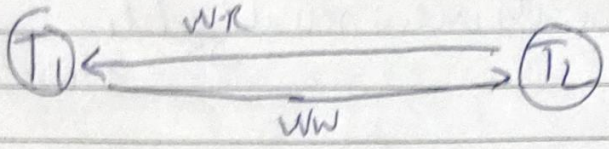
In the remaining operations

⇒ $R_1(A)\ W_1(A)\ W_2(A)\ W_3(A)$

Qn2) $R_2(B)\ W_2(A)\ R_1(A)\ R_2(A)\ W_1(B)\ W_2(B)$
$W_3(A)$



Q1.2) $R_2(B)\ W_2(A)\ R_1(A)\ R_3(A)\ W_1(B)\ W_2(B)\ W_3(B)$



T_2

not

Blind writes so conflict serializable

step 2:

	A		B
Initial Read	~~X~~		~~T2~~
Final Update	~~T2~~		~~T3~~
all write	~~T1~~ ~~T3~~ Random	T3 only write	T1 T2 T3 ↳ same as given read should be first

Views $\Rightarrow$ $R_2(B)\ W_2(A)\ W_2(B)\ R_1(A)\ W_1(B)$ ~~$R_3(A) W_3(A)$~~

A: $T_2 \Rightarrow$

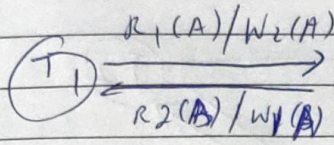
B: $T_2\ T_1\ T_3$ ~~$\Rightarrow$ (B)~~

View order follow B as A needs no order:

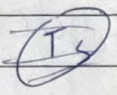
$R_2(B)\ W_2(A)\ W_2(B)\ R_1(A)\ W_1(B)$
$R_3(A)\ W_3(A)$

3) S: $R_1(A)\ R_2(A)\ W_2(A)\ R_2(B)\ W_1(A)\ R_1(B)\ W_1(B)$

Step 1: Precedence



There cycle present



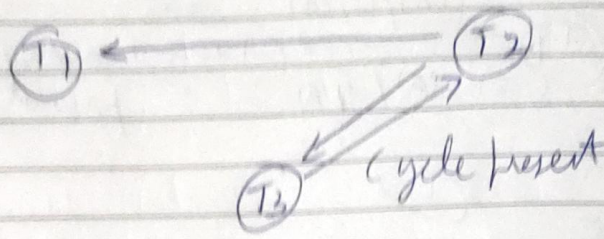
Step 2:

No change before R_1 reads

	A	B
Initial	T_1 T_2 serializable as non conflict	T_2 T_1
Final Update	T_1	T_1
All write	T_2 T_1	T_2 T_1

A: T_2 & T_1 } Same order
B: T_2 T_1 }

4) $R_1(A)\ R_2(A)\ R_3(B)\ W_1(A)\ W_2(C)\ R_2(B)\ W_2(B)\ W_3(C)$



	A	B	C
Initial Read	T_1, T_2	T_3, T_2	T_2
Final Update	T_1	T_2	T_3
All writes	T_1	T_2	T_3

A: $T_3 \rightarrow T_1$ $T_2 \rightarrow X \rightarrow T_1$

B: $T_3 \rightarrow Y \rightarrow T_2$

C: $T_2 \rightarrow T_3 \rightarrow T_3$

} ordering conflicting

So view serializable schedule possible.

Recoverable Schedules

- If a transaction T_j reads a data item previously written by transaction T_i, then the commit operation of T_i appears before commit of T_j.

Cascading Rollback

- A single transaction failure leads to a series of transaction rollbacks.

Cascadeless Schedules

- For each pair of transactions T_i and T_j such that T_j ~~reads~~ reads a data item ~~he~~ previously written by T_i, the commit operation T_i appears before the read operation T_j

Levels of Consistency in SQL 92

- Serializable - default
- Repeatable Read - Only committed records to be read, repeated reads must return same value
- Read Committed - Only committed records can be read but successive reads of record may return different
- Read Uncommitted - Even uncommitted records may be read.

Concurrency Controls

Lock based Protocol

Lock mechanism:
exclusive (X) for write operation
shared (S) for Read operation

lock – X instruction
lock – S instruction

Lock compatibility matrix

T_1 \ T_2	S	X
S	True	False
X	False	False

- Any number of transaction can hold shared locks on an item
- If any transaction holds an exclusive on the item no other transaction may hold any lock on the item

Two Phase locking Protocol

Phase 1: Growing phase
→ Transaction may obtain locks

Phase 2: Shrinking phase
→ Transaction may release locks.

Protocol ensures serializability.

Cascading Roll Back solution

i) Strict two-phase locking:

Here a transaction must hold all its exclusive (ads X) locks till it commits/aborts. After commit it must unlock.

ii) Rigorous two-phase locking is even stricter. Here all locks (X and S) are held till commit/abort.

Deadlock - Multiple transactions asking for locks when another transaction has access.

Deadlock Handling

- Deadlock Prevention protocol ensures that system will never enter into a deadlock state

Strategies:

1) Wait-die scheme (non preemptive) (forced to abort)

Older transaction may wait for younger ones to release data item. Younger transactions never wait for older ones they are rolled back instead.

iii) Wound-wait-preemptive

- Older transaction wounds of younger transaction instead of waiting for it. Yaugh transaction may wait for older one.

iv) Suppose in a database T_1, T_2 and T_3 with timestamp 10, 20, 30 respectively T_2 is holding a data item which T_1 and T_3 are requesting to acquire. Outcome with respect to wound wait

T_1 → continue
T_2 → Aborted
T_3 → wait for T_2

wait wait-die

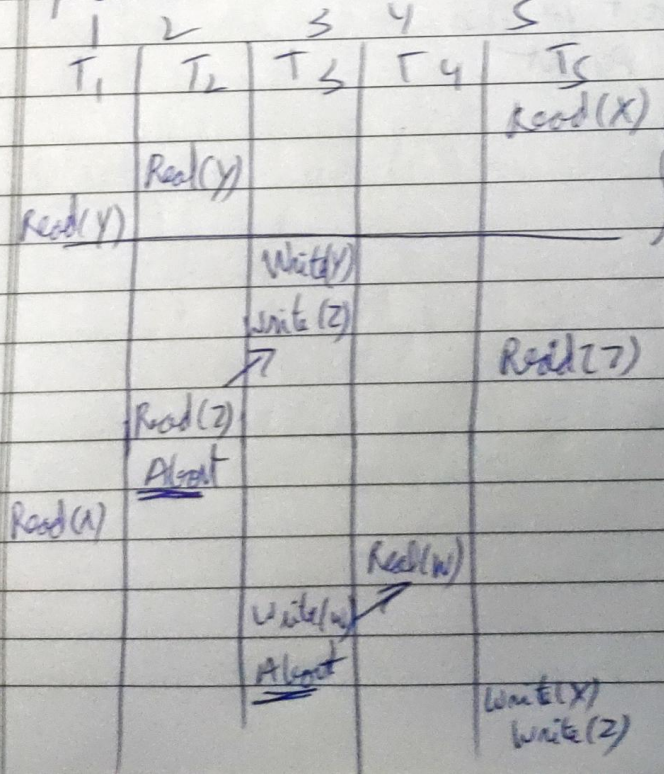
T_1 → wait for T_2
T_3 → Abort

Timestamp-Based protocols

The protocol maintains for each data 'Q' two time stamp values:

- W-timestamp(Q) is the largest time-stamp of any transaction that executed write(Q) successfully.
- R-timestamp(Q) is the largest time-stamp of any transaction that executed read(Q) successfully
- Timestamp ordering protocols ensure that any conflicting read and write operations are executed in timestamp order.

Ex: For transactions 1, 2, 3, 4, 5. Apply timestamp based protocol.



Aborted process:

$R_2(z)$ and $W_3(z)$
↳ Conflicting instructions after performing read

Transaction with smaller timestamp → Transaction with larger timestamp

Validation-Based Protocol (Optimistic Concurrency control)

Execution in 3 phases –

1) Read and execution phases: Transaction T_i writes only to temporary local variables.

2) Validation Phase – Transaction T_i performs a 'validation test' to determine if local variables can be written without violating serializability

3) Write phase – If T_i is validated all of data are applied otherwise rolled back.

Validation Phase:

$TS(T_i) < TS(T_j)$ either one of the following condition holds:

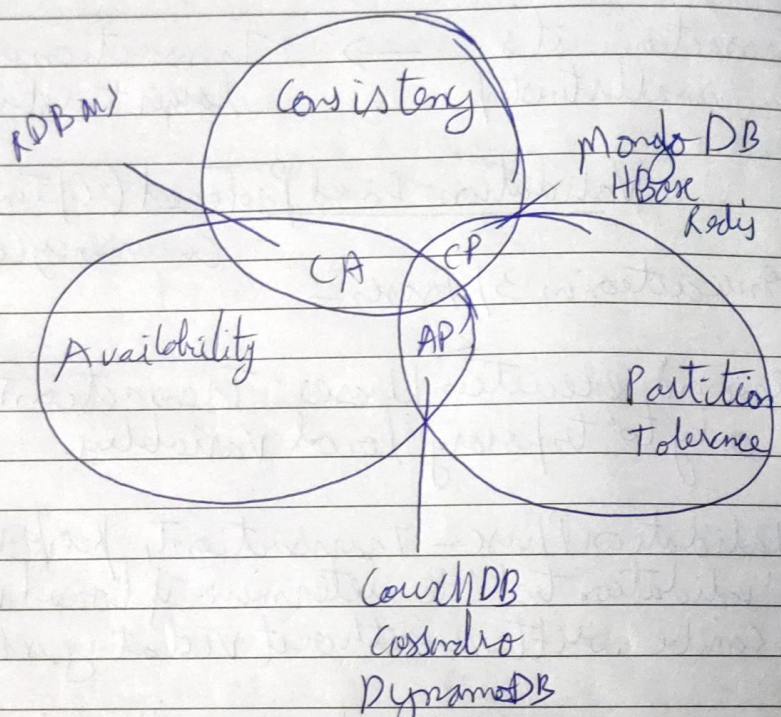
i) $finish(T_i) < start(T_j)$

ii) $finish(T_i) < validation(T_j)$ and set of data items written by T_i doesn't intersect with set of data items read by T_j or $validation(T_i) < finish(T_j)$

NoSQL

- Suitable for large amount of data, structured, unstructured or semi-structured.

Cap Theorem



In NoSQL P has to be there, Need to select either C or A.

dropA - Accept waiting until data is consistent

dropC - Accept getting inconsistent data

BASE: Basically Available, Soft State, Eventual Consistency.

BA: Use replication and sharding to reduce likelihood of data unavailability

S: Soft state indicates that state of system may change over time even without input.

E: Although application must deal with instantaneous consistency, NoSQL system assures that at some future point data assumes a consistent state.

Types:
- Key Value
- Column
- Graph
- Document

MongoDB

Open source document database designed using C++

- Each mongoDB instance can have zero or more databases each acting as high-level containers.
- A database can have zero or more collections
- A document is made up of one or more fields

Ex:

```
{
 "address": {
     "building": "1007",
        .
        .
 },
 "borough": "Bronx",
 "grades" [ ] → Multiple values
   {"date": {"$date": 1353124800}, "grade": "A",
        "score": 2},
        .
        .
 ];
```

Creation of database -> Use databasename

db.createCollection(name, options)
db.createCollection("mycol", capped: true, ...)
autoIndexID: true, size: 6142800, max: 10000)

db.restaurants.find() -> Display all.

db.restaurants.find({"borough": "Bronx"}).limit(5)

db.restaurant.find({"borough": "Bronx"}, {"cuisine": 1}).limit(15).sort({"cuisine": 1})

displays only cuisine

sort cuisine based on alphabets

ASC

-1 = DSC

Query to find restaurants with score more than 90

db.restaurant.find({grades: {$elemMatch: {"score": {$gt: 90}}}});

Elements from

Find where cuisine not American

db.restaurants.find({"cuisine": {$ne: "American"}})