Database Management System

Data - Raw and isolated facts about an entity (recorded) like text, audio, image, etc.

Information - Processed, meaningful and usable data

Database - Collection of similar/related data

DBMS - Software used to create, manipulate and delete database.

Disadvantages of File System:

(i) Data redundancy
(ii) Data inconsistency
(iii) Difficulty in accessing data
(iv) Data isolation
(v) Security Problem
(vi) Atomicity Problem
(vii) Concurrent-access anomalies
(viii) Integrity Problem

P.T.O →

Comparison between two different types of databases:

OLAP (Online Analytical Processing)	OLTP (Online Transaction Processing)
(i) Historical data	(i) Current data
(ii) Subject oriented	(ii) Application oriented
(iii) Decision making	(iii) Day to day operations
(iv) Larger size of data	(iv) Smaller size of data
(v) Only read operations required	(v) Read and write, both operations required

Entity Relationship Diagram: It was introduced by Dr. Peter Chen in 1976. It is a non-technical design method working on conceptual level based on the perception of real world.

Phone/Email/Notes

Notes • Consists of ~~collections~~ collection of basic objects called entities and of relationships among these objects and attributes which defines their properties.

- Free from ambiguities and provides a standard and logical way of visualizing data.
- Basically, it is a digrammatic representation and it is easy to understand even by non-technical user.

Entity: An entity is a thing or an object in the real world that is distinguishable from other objects / things based on the values of the attributes, it ~~possesses~~ possesses.

Types of Entities:

① Tangible - Entities which ~~ex~~ physically exist in real world. Eg., car, pen, etc.

② Intangible - Entities which exists logically, Eg., bank account, ~~etc~~ etc.

Entity Set: Collection of same type of entities i.e. those share the same properties or attributes

P.T.O →

Entity cannot be represented in an ER diagram as it is an instance / data. It can be represented in a relational model by row / tuple / record.

Entity set is represented by rectangle in ER diagram and is represented by table in a relational model.

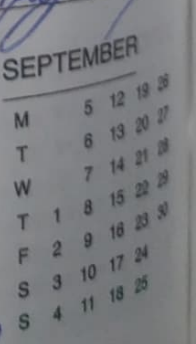
Attributes: These are the units that describe the characteristics of entities.

- For each attribute, there is a set of permitted values called domains.
- In ER diagram, represented by ellipse or oval, while in relational model by a seperate column.

Types:- (i) Simple-Composite: Simple, cannot be divided further & is simply representated by an oval.

(ii) Single multivalued: It can have only one value at an instance of time.

Composite can be further divided in simple attribute where one oval can be connected with another.



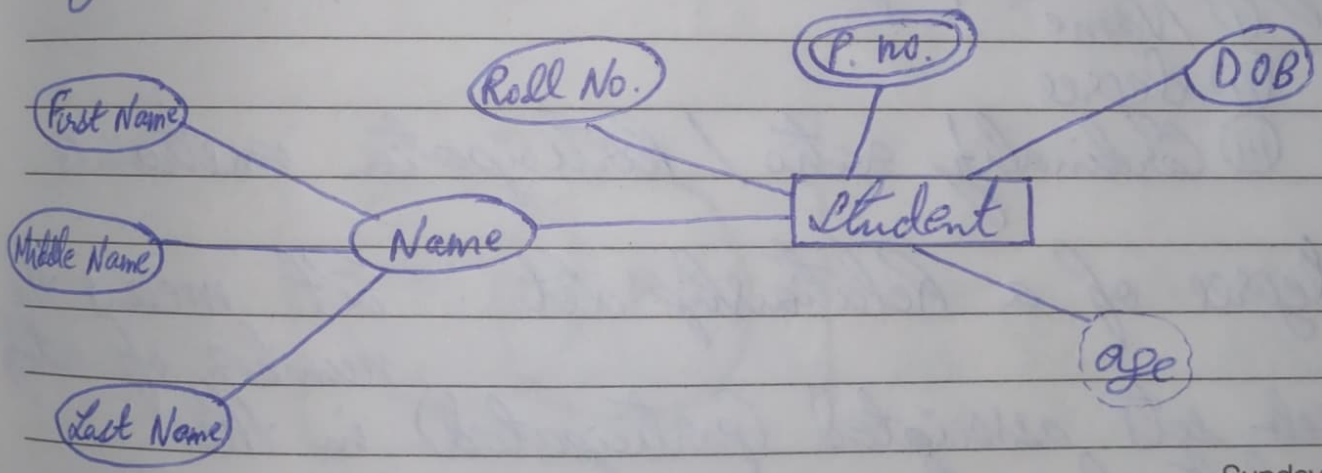
Single

(ii) ~~Simple~~ - Multivalued : Single can have only one value at an ~~ins~~ instance of time.

Multi Multivalued can have more than one value at an ~~ins~~ instance of time.

(iii) Store - Derived : Stored is how value is stored in database.

Derived is how value can be computed in runtime using stored attribute



Sunday 09

Relationship : It is an association between two or more entities of same or ~~different~~ different entity set.

It has no representation in ER diagram as it is an instance or a data.

In relational model, represented using

10/ Monday

a row in a table.

Relationship type / set : A set of similar types of relationship.

Represented using a ~~set~~ diamond in ER diagram. In relational model, either by a seperate table or by a seperate column. (foreign key)

Every relationship type has three components -
(i) Name
(ii) degree
(iii) Cardinality ratio / participation constraints

Degree of a Relationship set : It means the number of entity ~~sets~~ sets associated (participated) in the relationship set.

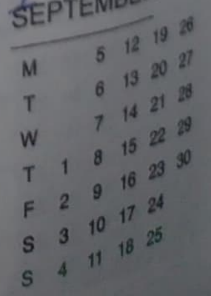
Most of the relationship sets in the E-R diagram are binary. Occasionally, however relationship sets involve more than two entity sets.

Phone/Email/Notes

Min. degree of relationship set = 1

Notes

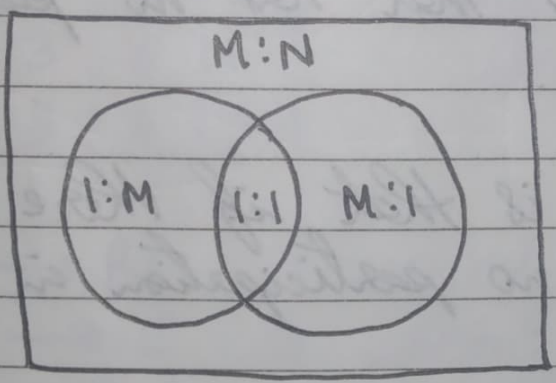
P.T.O →



Mapping Cardinalities / Cardinality Ratio:

It expresses the number of entities to which other entity can be related via a relationship. It can be used in describing relationship set of any degree but is most useful in binary relationship.

1:1 (One to one)
1:M (One to many)
M:1 (Many to one)
M:N (Many to many)



Participation Constraints: Specifies whether the existance of an entity depends on its relation to another entity via a relationship type.

These constraints specify the max and min no. of relationship instances that each entity can/must participate in.

Max Cardinality : It defines the max number of times, an entity occurrence participates in a relationship.

Min Cardinality : It defines the min no. of times, an entity occurrence participates in a relationship.

Notation : (min c, max c)

Partial participation is that if there exists atleast one entity that has no participation in the relationship.

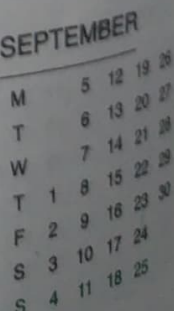
Total participation is that if there exists no entity which has no participation in the ~~not~~ relationship.

Functional Dependencies :

	α	β
$t_1 \rightarrow$	a	1
$t_2 \rightarrow$	a	1
	c	2
	d	3

$\beta : \alpha \rightarrow \beta$ Here, using the value of α, we can map it in the table and search for the value of β.

If $t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta]$



α is called determinant & β is called dependent.

$$f: \alpha \longrightarrow \beta$$

Trivial
$\beta \subseteq \alpha$

Non-trivial
$\beta \not\subseteq \alpha$

Q:)

R

A	B	C	D	E
a	2	3	4	5
2	a	3	4	5
a	2	3	6	5
a	2	3	6	6

ⓐ $A \longrightarrow BC$
ⓑ $DE \longrightarrow C$
ⓒ $C \longrightarrow DE$
ⓓ $BC \longrightarrow A$

Sol: ⓐ $A \longrightarrow BC$ is correct
ⓑ $DE \longrightarrow C$ is correct
ⓓ $BC \longrightarrow A$ is correct

We check if on same value of α, we get two different values of β. If so, then the functional dependency is not valid. Otherwise, the functional dependency is valid.

* If all the values of α are unique, f.d. is always valid.
* If all the values of β are same, f.d. is always valid.

Attribute Closure / Closure on Attribute Set / Closure of Attribute Set :

Attribute Closure of an attribute set 'A' can be defined as a set of attributes which can be functionally determined from it. Denoted by F^+.

R(ABCDEF) : $A \rightarrow B$, $C \rightarrow DE$, $AC \rightarrow F$,
$D \rightarrow AF$, $E \rightarrow CF$

From above, $(D)^+ = ABDF$, $(DE)^+ = ABCDEF$

Armstrong Axioms/Rules : These hold true for relational databases & can be used to generate closure set.

Primary Rules - Reflexivity : If $Y \subseteq X$, then $X \rightarrow Y$

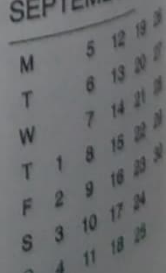
Augmentation : If $X \rightarrow Y$, then $XZ \rightarrow YZ$

Transitivity : If $X \rightarrow Y$ && $Y \rightarrow Z$, then $X \rightarrow Z$

Phone/Email/Notes

Notes

Secondary Rules - Union : If $X \rightarrow Y$ && $Y \rightarrow Z$, then $X \rightarrow YZ$



Decomposition: If $X \to YZ$, then $X \to Y$ && $X \to Z$

Pseudo Transitivity: If $X \to Y$ && $WY \to Z$, then $WX \to Z$

Composition: If $X \to Y$ && $Z \to W$, then $XZ \to YW$

Closure Set of Attributes:

R(ABC) $A \to B$ $B \to C$

Let $F = f_1 + f_2$ $\therefore A \to BC$

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

Equivalence of Functional Dependency:

Sunday 16

R(ACDEH) F: $A \to C$, $AC \to D$, $E \to AD$, $E \to H$

G: $A \to CD$, $E \to AH$

(a) $F \subseteq G$ (b) $F \supseteq G$

(c) $F = G$ (d) $F \neq G$

Sol: P.T.O →

Finding closure of LHS of F using G

$(A)^+ = ACD$
$(AC)^+ = ACD$
$(E)^+ = EAHCD$

Comparing this with the functional dependencies of F

∴ All the dependencies of G can be done by F
∴ $F \subseteq G$

Finding closure of L.H.S of G using F

$(A)^+ = ACD$
$(E)^+ = EADHC$

Comparing this with the f.d. of G

∴ All the dependencies of F can be done by G
∴ $G \subseteq F$

∴ $F = G$ //

Irreducible Let of Functional Dependency (Cannonical form):

$R(WXYZ)$ $X \rightarrow W$
$WZ \rightarrow XY$
$Y \rightarrow WXZ$

Decomposing these f.d., we have

Phone/Email/Notes
Notes

✓ $X \rightarrow W$
$XWZ \rightarrow X$
✓ $WZ \rightarrow Y$
$XY \rightarrow W$
✓ $Y \rightarrow X$
✓ $Y \rightarrow Z$

$(X)^+ = XW$
$(WZ)^+ = WZXY$
$(Y)^+ = YWXZ$

$(X)^+ = X$
$(WZ)^+ = WZYX$
$(WZ)^+ = WZ$
$(Y)^+ = YXZW$
$(Y)^+ = YZ$
$(Y)^+ = YXW$

After filtering, we have

$X \rightarrow W$, $WZ \rightarrow Y$, $Y \rightarrow X$, $Y \rightarrow Z$

Checking the 'x' values of f.d.,

$(WZ)^+ = WZYX$
$(W)^+ = W$ $(Z)^+ = Z$

~~Canonical~~ Cannonical Form : $X \rightarrow W$
$WZ \rightarrow Y$
$Y \rightarrow XZ$

* There can be different minimal forms for a same set of f.d.

Keys: When we find the closure set of key, we get the complete relation

$(Key)^+ = R$

It is a set of attributes which can uniquely identify a row or a tuple in a relational table.

(i) R(ABCD) ; $A \longrightarrow BC$

Here, A is not key as $(A)^+ = ABC \neq R$

(ii) R(ABCD) ; $ABC \longrightarrow D$, $AB \longrightarrow CD$, $A \longrightarrow BCD$

Here, $(ABC)^+ = (AB)^+ = (A)^+ = R$. Thus, all are keys

Super Key : It is a set of attributes with the help of which you can uniquely identify ~~a row~~ all the remaining attributes. This is the most basic key.

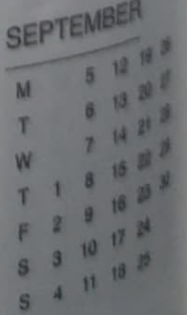
Candidate Key : A super key is candidate key whose no proper subset is a super key again.

(iii) R(ABCD) ; $B \longrightarrow ACD$, $ACD \longrightarrow B$

Phone/Email/Notes

Notes Here, $(B)^+ = (ACD)^+ = R$

Both B and ACD are candidate keys.



Primary Key: It is that candidate key which is selected by a database administrator as a primary way to identify tuples.

(iv) R(ABCD) ; AB → C , C → BD , D → A

$(AB)^+ = ABCD = R$, $(C)^+ = CBDA = R$, $(D)^+ = DA$

Here, AB, C are ~~candidate~~ super keys and candidate keys.

Normalization :

(i) First Normal Form (1NF) – Every cell should have a single value.

If there is a multivalued attribute present, we duplicate all the other attributes and convert it into 1NF.

In general, for every multivalued attribute, we make a seperate table.

Here, the order of rows and the order of columns is irrelevant. Also, every column should have a unique name.

(ii) Second Normal Form (2NF) – R(ABCD) $(AB)^+ = ABCD = R$

Here, A, B are prime attribute and others are non prime attribute.

~~Here, C is only dep~~ When a non-prime attribute instead of depending on the entire candidate key, is dependent on a part of it is called partial ~~depend~~ dependency.

Thus, for a table to be in 2NF, it should be in 1NF and it shouldn't have any partial ~~depe~~ dependency.

We decompose R(ABCD) into two ~~tables~~ tables

$R_1(ABD)$, $R_2(BC)$

One table is for the ~~cand~~ complete candidate key and all the ~~complete~~ attributes ~~have~~ having no partial dependencies.

Another table is for the partial dependency.

Thus, we obtain 2NF

Phone/Email/Notes

Notes

(ii) Third Normal form (3NF) – A functional dependency from $\alpha \to \beta$ is called ~~trans~~ transitive if α, β are non prime.

$R(ABC)$; $A \to B$, $B \to C$

A relation R is in ~~the~~ 3NF if it is in 2NF and there should not be transitive dependency.

Also if every dependency from $\alpha \to \beta$ where α is a Super Key or β is a prime attribute, then it is in 3NF.

(iii) Boycecodd Normal form (BCNF) – It deals with $\underset{P/N.P}{\alpha} \to \underset{P}{\beta}$

$R(ABC)$; $(AB)^+ = ABC = R$ $\quad AB \to C$
$(AC)^+ = ABC = R$ $\quad C \to B$

Sunday 23

Both the f.d. are neither partial nor transitive. Thus, the table is in 2NF and 3NF.

~~For α~~ For the table to be in BCNF, α should be super key.

Normalising the table, R(ABC)

$R_1(CB)$ $R_2(AC)$

Ways to check normal form:

(i) For every dependency from $\alpha \rightarrow \beta$, if α is super key, then it is BCNF.

(ii) For every dependency from $\alpha \rightarrow \beta$, if α is ~~candidate~~ super key, or β is ~~non~~ prime, then it is 3NF.

(iii) For every dependency from $\alpha \rightarrow \beta$, ~~if β is α not~~ & is C.K. is not non prime, then it is 2NF.

(iii) For every dependency from $\alpha \rightarrow \beta$, if α is C.K. or β is prime, then it is 2NF.

Lossless join decomposition / non-additive:

This property guarantees that any extra or less row/tuple generation problem does not occur after decomposition.

- It is a mandatory ~~property~~ property & ~~must~~ must always hold good.

Phone/Email/Notes

Notes

- If a relation R is decomposed into two relations R_1 & R_2, then it will be lossless if and only if
  (i) $attr(R_1) \cup attr(R_2) = attr(R)$
  (ii) $attr(R_1) \cap attr(R_2) \neq \emptyset$
  (iii) $attr(R_1) \cap attr(R_2) \rightarrow attr(R_1)$
  or
  $attr(R_1) \cap attr(R_2) \rightarrow attr(R_2)$

Dependency Preservation: If a table R having f.d. set F is decomposed into two tables R_1 and R_2 having f.d. set F_1 & F_2 then $F_1 \subseteq F^+$, $F_2 \subseteq F^+$ i.e. $(F_1 \cup F_2) = F^+$. This means the decomposition is dependency preserving.

R(ABCD) ; $AB \rightarrow CD$, $D \rightarrow A$

$R_1(AD)$	$R_2(BCD)$
F_1: $D \rightarrow A$ ✓	F_2: $B \rightarrow A$ ✗
$A \rightarrow D$ ✗	$CD \rightarrow CDA$ ✗
	$BD \rightarrow BDAC$ ✓

The dependency $AB \rightarrow CD$ is lost.
Dependency preservation is not compulsory while table decomposition.

* Those attributes which donot have any incoming edge are called ~~essential~~ essential attributes.

10 / 28 / Friday / Wk-44/Day (302-064)

Normalisation of a Table: R(ABCDEFGHIJ)

$(AB)^+ = ABDECFGHIJ \rightarrow$ C.K.

The current R is in 1NF. We normalise it further

$R_1(ABC)$, $R_2(ADEIJ)$, $R_3(BFGH)$

Current schema is in 2NF. We normalise it further

$R_1(ABC)$, $R_{21}(ADE)$, $R_{22}(DIJ)$, $R_{31}(BF)$, $R_{32}(FGH)$

Current schema is in 3NF & BCNF as well.

Database Management System

File Structures:

Sorted File	Unsorted File
1) Can be sorted only according to ~~one~~ ~~alternate~~ attribute (search key).	1) Random order, any record can be placed anywhere.
2) Searching is fast (binary search).	2) ~~Search is~~ Searching is slow (~~bin~~ linear search).
3) Insertion and deletion can be difficult.	3) Insertion and deletion can be easy.

Spanned / Unspanned Mapping:

Spanned is when a table is broken into parts while storing if the block cannot store a complete table. Costly in time.

~~Un~~ Unspanned is when a table ~~of~~ is stored in a block if and only if the block a can store a complete table. Costly in space.

Indexing: We make a file called as the index file. We store two columns, Search Attribute and Block pointer.

* The block pointer ~~cant~~ contains the base address ~~of the who~~ of that block which contains the ~~sea~~ search attribute.

If selected records get stored in the index file, then it is known known as sparse indexing.

If all values ~~records~~ get stored in the index file, then it is

known as dense indexing.

Types of Indexing:

We can make multiple index files depending upon the size of the index file. It is known as ~~multi~~ multi-level indexing.

(1) Primary Indexing - Here, the search attribute is the primary key of the table. It is done when the main file is sorted.

(2) Clustered Indexing - Here, the search attribute is not a key of the table.

(3) Secondary Indexing - Here, the search attribute is either key or non-key and the main file is not sorted.

* Index file will always be sorted.

Primary Indexing: (i) The main file is sorted.

(ii) Primary key is used as search attribute.

(iii) It is an example of sparse indexing.

(iv) No. of entries in the index file = No. of blocks acquired by the main file.

(v) No. of accesses required = $\log_2 n + 1$

Let ~~us~~ us assume a block size of ~~1024 MB~~ 1024B.

Blocking factor = $\left\lfloor \frac{\text{Block size}}{\text{Record size}} \right\rfloor = \left\lfloor \frac{1024B}{15B} \right\rfloor$ ~~= [68.26]~~

$= \lfloor 68.26 \rfloor = 68$

No. of blocks for index file = $\left\lceil \frac{3000}{68} \right\rceil = 45$

$\therefore$ No. of block access required $= \lceil \log_2 45 \rceil + 1 = 6 + 1 = 7$

Clustered Indexing : (i) The main file is sorted (on a non-key attribute).

(ii) There will be only one entry for each unique value of the non-key attribute.

(iii) If the number of blocks acquired by index file is n, then block access required will be $\geq \log_2 n + 1$.

Secondary Indexing : (i) The main file is unsorted.

(ii) It can be done on key as well as non-key attributes.

(iii) It is called secondary because normally one indexing is already done

(iv) It is an example of dense indexing, no. of entries in index file = no. of entries in main file.

(v) No. of block access $= \lceil \log_2 n \rceil + 1$

Let us consider a block size of 1024 B. The record size in main file is 100 B and in index file is 15 B. The no. of records in main file are 30000.

$$\text{Blocking factor} = \left\lfloor \frac{\text{Block size}}{\text{Record size}} \right\rfloor = \left\lfloor \frac{1024\,B}{15\,B} \right\rfloor = 68$$

$$\text{No. of blocks for index file} = \left\lceil \frac{\text{No. of records for index file}}{\text{Blocking factor}} \right\rceil = \left\lceil \frac{30000}{68} \right\rceil = 442$$

$\therefore$ No. of block access required $= \lceil \log_2 n \rceil + 1 = 9 + 1 = 10$

Transactions : It is a set of instructions which perform a logical unit of work. They are atomic in nature.

ACID Properties :

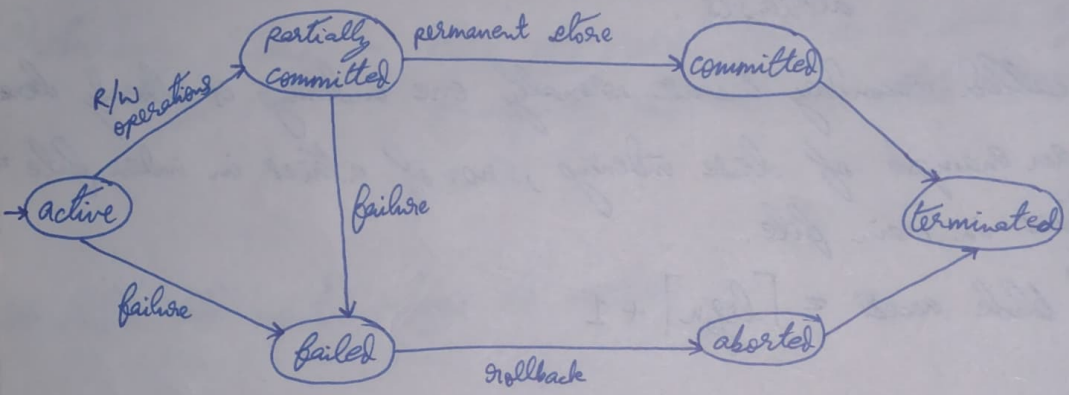
(i) Atomicity :- It means the transactions are atomic in nature.

(ii) Consistency : After the transaction is completed, the DB should be consistent.

(iii) Isolation : Each transaction should be logically isolated of the other.

(iv) Durability : The transaction carried out should be durable in nature.

Transaction States :



* A committed transaction cannot be rolled deployed.
* Concurrency control component takes care of isolation.
* Recovery manegement component takes care of durability.

Advantages of Concurrency :

(i) Waiting time↓ - After a process is ready, the time it waits until execution starts is known as waiting time.

(ii) Response time↓ - The time taken by the first response of the CPU after the process is ready is known as ~~response~~ response time.

(iii) Resource Utilization↑ - It is the ~~ut~~ utilization of the resources

for carrying out the concurrent tasks.

(iii) Efficiency ↑ - It determines how much resources ~~you~~ have been used for carrying out the concurrent tasks.

~~Efficiency = No. of tasks / No. of resources × 100 %~~

t. Dirty Read problem : In case of concurrent transactions, when one transaction reads data from the local buffer of another transaction, and ~~commits~~ commits it, then it is known as ~~the~~ dirty read. The problem occurs when there is a failure in the other transaction. This leads to rollback of the transaction and the ~~value~~ data read and committed by the first transaction ~~doesn't~~ doesn't exist in the ~~database~~ database anymore. This leads to inconsistency.

* To solve dirty read problem, the first transaction should only commit after the second transaction has committed. This allows the first transaction to rollback if necessary.

Unrepeatable Read problem : In case of concurrent transactions, when there are multiple reads of data by some transaction and it ~~also~~ gives different output each time, then it is ~~how~~ known as unrepeatable read problem.

Phantom Read problem : In case of concurrent transactions, when there are multiple reads of data by some transaction and it is unable to read data at some point, then it is known as phantom read problem.

Lost Update problem (Write-Write conflict) : In case of concurrent transactions, when there

are multiple transactions writing to the same database, the data ~~upd~~ updated by the previous transactions get lost as the data gets ~~w~~ changed by other transactions and finally committed. This is known as lost update problem.

Schedule : It is used to schedule concurrent transactions.

Serial Schedule - Transactions are executed one after the other.

Non serial Schedule - Transactions are switched among one another and are executed.

* For n transactions, n! serial schedules are possible.

* n n n , $\frac{(n_1+n_2+\ldots+n_n)!}{n_1!.n_2!\ldots n_n!} - n!$ non serial schedules are possible.

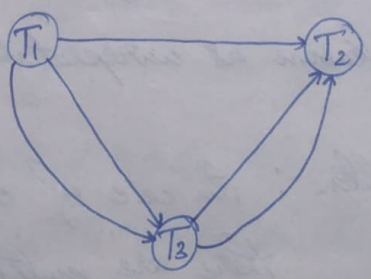
Conflict Serializibility : We use this to convert a non serial ~~sch~~ schedule to a serial ~~ser~~ schedule.

* Conflict ~~occ~~ occurs when there are write instructions ~~R & W operations~~ on the same data. ~~of the transactions~~ concurrent transactions.

Q:)

T_1	T_2	T_3
× R(X)		
	× R(Y)	
		× R(Y)
	× W(Y)	
× W(X)		
		× W(X)
	× R(X)	
	× W(X)	

Sol:



There are no cycles in this graph. Thus, ~~the~~ the given schedule is conflict serializable.

There are no incoming ~~each~~ edge to T_1. Thus, T_1 will run first.

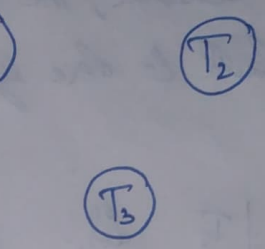
There are no outgoing edges from T_2. ~~Three~~ Thus, T_2 will run last.

∴ Order ~~of~~ of serializability : $T_1 \rightarrow T_3 \rightarrow T_2$

Q:)

T_1	T_2	T_3
× R(a)		
	× R(b)	
		× R(c)
		× W(c)
	× W(b)	
× W(a)		

Ans:



There are no cycles present in the graph. Thus, the schedule is conflict serializable.

∴ No. of serial schedules possible = 3! = 6

* $O(n^2)$ is the time complexity for finding out ~~who~~ whether a schedule is conflict serializable or not.

View serializability : It is a superset of conflict serializability. A schedule is not conflict serializable but can be ~~view~~ view serializable when there is atleast one blind ~~write~~ write present in the schedule.

When the given non-serial schedule is view equivalent to any one of the n! serial schedules, then the given schedule is view ~~serial~~ serializable.

* Conditions for View Equivalent :

(i) The transaction which does the initial read on a data in first schedule, ~~should do be same~~ that transaction only ~~show~~ should do the initial read on that data in second schedule.

(ii) The transaction which does the final write on a data in first

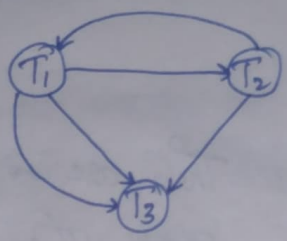
schedule, that transaction only should do the final write on that data in second schedule.

(iii) In first schedule, the intermediate reads done by a transaction should be done by the same transaction in the second schedule.

Q:)

T_1	T_2	T_3
R(a)		
	W(a)	
W(a)		
		W(a)

Sol:



~~The go~~ There is a cycle present. So, the schedule is conflict serializable.

Now, there is blind write present in T_2 and T_3. Thus, the schedule ~~s~~ can be view serializable.

Taking a serial schedule,

T_1	T_2	T_3
R(a)		
W(a)		
	W(a)	
		W(a)

Comparing this with the given non serial schedule, we can observe that all the three conditions of ~~is~~ view equivalent is satisfied.

Thus, the given non-serial schedule is view-serializable.

Recoverable Schedule : These schedules can handle failures in transactions and ensure that the consistency is maintained.

* If there no dirty read present in the schedule, then it is a recoverable schedule.

* If there are dirty reads present in a schedule, then the transaction doing the dirty read should commit later to make it a recoverable schedule.

* A schedule should always be recoverable.

Cascadeless Schedule :

When multiple transactions rollback one after the other due to a failure, then it is known as cascading rollbacks.

When there are no cascading ~~roll~~ rollbacks occuring in a schedule, then it is ~~kno~~ known as cascadeless schedule. To make a cascadeless schedule, we remove dirty reads from the schedule.

Strict Schedule : In a strict schedule, there are no dirty reads by any other transaction as well as no write operation can be performed until the first ~~original~~ transaction commits the data.

Eg.,

S_1

T_1	T_2
R(a)	
W(a)	
C ✓	
	R(a) W(a)
	R(a)
	C

S_2

T_1	T_2
R(a)	
	R(b)
W(a)	
	W(b)
C ✓	
	R(a)
	C

Concurrency Control Technique :

The main problem is different transactions trying to access data at the same time.

For that we use, ① Time stamping Protocol

(ii) Lock based protocol

- 2PL (Basic, Conservative, Strict, Rigorous)
- Graph based

(iii) Validation protocol

Time stamping protocol : The basic idea is to decide the order between the transactions before that enters into the system so that in case of conflict during execution, we can resolve the conflict using ordering.

The reason we call it time stamp and not just stamp because for stamping we use the value of the clock as it is always unique and never repeat itself.

Two ways of time stamping -

(i) Time stamp with transaction : With each transaction t_i, we associate a time stamp denoted by $T.S.(T_i)$, it is the value of the system clock when a transaction enters into the system so if a new transaction T_j, enters after T_i then, $T.S.(T_i) < T.S.(T_j)$ which will always be unique and will remain fixed throughout the execution. It also determines the serializability order if $T.S.(T_i) < T.S.(T_j)$ then system ensures that the resultant conflict serializable schedule T_i will execute first, before T_j executes.

(ii) Time stamp of each data item : For each data item Q, the protocol maintains ~~two~~ two time stamps.

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.

Ti request for Read (Q) :

If T.S. (Ti) < W.T.S. (Q), means Ti needs to read a value of Q that was already overwritten. Hence, request is rejected and Ti must rollback.

If T.S. (Ti) ≥ W.T.S. (Q), operation can be allowed and R.T.S. (Q) will be max (R.T.S. (Q), T.S. (Ti)).

Ti request for Write (Q) :

If T.S. (Ti) < R.T.S. (Q), means value of Q that Ti is producing was needed previously and the system assumed that the value would never be produced and hence rejected and rolled back.

If T.S. (Ti) < W.T.S. (Q), Ti is attempting to write an obsolete value of Q, thus Ti is rejected and rolled back otherwise it is allowed. W.T.S. (Q) = max (W.T.S. (Q), T.S. (Ti)).

Properties of Time Stamping Protocol -

(i) It ensures conflict serializability.

(ii) It ensures view serializability.

(iii) Possibility of dirty reads and no restriction on commit, so irrecoverable schedules and cascading rollbacks are possible.

(iv) No deadlock as we either allow or reject the transaction.

(v) Time stamp schedules made by time stamping are a subset of conflict serializable schedule.

(iv) It may suffer from starvation and is relatively slow.

Thomas Write Rule : Modifies time stamping protocol to make some improvements which may generate those ~~sch~~ schedules that are V.S. but not C.S. and provides better concurrency.

It modifies time stamping protocol in ~~obsolete~~ blind write case when T_i request W(Q) if ~~T.S.~~ T.S. $(T_i) <$ W.T.S.(Q).

Here, T_i attempts to write obsolete value of Q so rather than rolling back T_i, write ~~operation~~ operation is ignored.

Lock Based Protocol : To achieve ~~consis~~ consistency, isolation is the most important idea and locking is used to achieve isolation. First obtain a lock on a data item, then perform the ~~desired~~ desired operation and then unlock it.

Different modes of locks -

(i) Shared mode : It is denoted by lock-S(Q). Transaction ~~that~~ can perform ~~read~~ read operation and any other transaction can ~~also~~ also obtain the same lock on the same data item at the same time.

(ii) Exclusive mode : It is denoted by lock-X(Q). Transaction can perform both R&W operation and any other transaction cannot be obtained either with shared or exclusive lock.

~~Proper~~ Properties of Lock Based Protocol -

(i) If we do unlocking, inconsistency will arise and if we don't unlock, then concurrency will be poor.

(ii) The transaction should follow a set of rules for locking and

unlocking of data item like 2-Phase locking or graph based.

(iii) A schedule S is legal under a protocol if it can be generated using the rules of the protocol.

Two-Phase Locking (2PL): (Basic) (i) This protocol requires that each transaction in a ~~schedule~~ schedule will be ~~two~~ two phased, Growing phase and Shrinking phase.

(ii) * In growing growing phase, transaction can only obtain locks but cannot release any lock.

* In shrinking phase, transaction ~~can~~ can only ~~obt~~ release locks but ~~cannot~~ cannot ~~obtain~~ obtain any lock.

Transaction can perform both R&W operations, both in growing and shrinking ~~ph~~ phase. It ensures C.S. and V.S. and the order of serializability is the order in which transaction reaches the lock point. It may generate ~~in irrec~~ irrecoverable schedules and cascading rollbacks and does not ~~et~~ ensure freedom from deadlock.

Conservative/Static 2PL: There is no growing phase, in the transaction, first we will acquire all the locks ~~requir~~ required and then directly will start from the lock point.

If all the locks are not available, then the transaction must release the lock acquired so far and wait. Shrinking phase works as usual ~~and~~ and the transaction can unlock any data item.

* C.S., V.S. and indpendant of deadlock but possibility of ~~irrecoverab~~ irrecoverable schedules and cascading rollbacks.

Rigorous 2PL: It is an improvement over 2PL protocol where we try

to ensure recoverability and cascadelessness.

(i) It requires that all the locks must be held until the transaction commits i.e. there is no shrinking phase in the life of a transaction.

(ii) Will ensure C.S., V.S., recoverability and cascadelessness.

(iii) Suffers from deadlock and inefficiency.

Strict 2PL : It is an improvement over rigorous 2PL.

(i) In the shrinking phase, unlocking of exclusive task are not allowed but unlocking of shared tasks can be done.

(ii) All the properties are same as that of rigorous 2PL, but it provides better concurrency.

DBMS NOTES BY RISHIRAJ
REFERENCE : KNOWLEDGE GATE YOUTUBE

Relational Algebra: After designing of database, i.e. ER diagram design then converting it into relational model followed by normalization & indexing. Now, the task is how to store, retrieve and modify data in database. Though here, we will be concentrating more on the query part.

Query Language: Language in which the user requests some information from the database.

Procedural Query Language: Here, user instructs the system to perform a sequence of operations in order to produce desired result. User tells what data to be retrieved & how to be retrieved.

Non-Procedural Query Language: Here, user describes the desired information without giving the specific procedure for obtaining that information.

In practice, we use RDBMS (practical implementation of relational model)

- SQL is used to write query on it.
- Relational model is a conceptual/theoritical framework & RDBMS is its implementation.
- Relational algebra (procedural) & Relational calculus (Non-procedural) are mathematical system or query language used on Relational Model.

Fundamental Operators: Relational algebra is one of the two formal query languages associated with relational algebra.

- Like any other mathematical system, it defines a no. of ~~operations~~ operators & use relations (tables) as operands.
- Every operator in relation algebra take one or two relations as input argument & generate a single relation as a result without a name.
- Relational Algebra doesnot consider duplicacy as it is based on set theory.
- In each query, we describe ~~stet~~ step by step ~~procedural~~ procedure for computing the ~~desu~~ desired output. So, it is a procedural query language.
- No use of English keywords.

Basic / Fundamental	Derived
Select (σ)	Natural Join ($\bowtie$)
Project (π)	$\bowtie$, $\bowtie$, $\bowtie$
Union ($\cup$)	Intersection ($\cap$)
Set difference ($-$)	Division ($\div$)
~~Hhh~~ Cartesian Product ($\times$)	
Rename (ρ)	

Select (σ): Select is a unary ~~operator~~ operator so ~~that~~ can take only one table at a time.

- It is a fundamental operator.
- Main idea of select is to find those tuples / rows in a relation which satisfies a given ~~set relation~~ condition.
- Syntax, $\sigma_{\text{condition/predicate}}(\text{table\_name})$

- It has same function as of 'where' clause in SQL.
- Min. no. of tuples selected is 0.
- Max. no. of tuples ~~el~~ selected is all the tuples.

Project (Π): Project is a unary operator, takes one table at a time.

- Fundamental operator.
- Main idea of project is to select desired columns.
- Syntax, $\Pi_{column\_name}(table\_name)$
- It works as 'select' clause of SQL.

Operators such as Union (∪), Intersection (∩) and Set Difference (−) are needed for multiple table queries.

Cartesian Product (X): It is a binary operator, takes two tables at a time.

- Fundamental Operator.
- Allows us to combine ~~two~~ information between two tables.
- $R_1(1, 2, 3, \ldots, n)$, $R_2(1, 2, 3, \ldots, m) \longrightarrow R_1 \times R_2(1, 2, 3, \ldots, m+n)$

After performing cartesian product, we get redundant (surplus) tuples.

Q:) Find customer name having account balance < 100.

Sol: $\Pi_{cust\_name}(\sigma_{Account.account\_no = Depositer.account\_no \wedge balance < 100}(Account \times Depositer))$

This removes the redundancy

- In relational algebra, cartesian product is a commutative & associative operator.

Introduction to SQL

- Number of query languages are in use.
- Structured Query Language, SQL is the most popular & widely accepted query language.
- It is domain specific & not general purpose, used in design and management of data held in DBMS.
- Can do much more than just a query on a database. Can define structure of database, modify data in database, specify security constraints & number of other tasks.
- Based on relational algebra and tuple relational calculus.

Parts of SQL:

(i) Data Definition Language (DDL) - It provides commands for defining relational schemas, deleting relations and modifying relational schemas.

Integrity - Provides commands for integrity constraints on the data stored in database. Eg., primary key cannot be null, foreign key reference, etc.

View Definition - Commands for defining views, eg. sorting the result according to some attribute.

Authorization - Commands for access rights, eg., read only, R/W grants, etc.

(ii) Data Manipulation Language (DML) - It provides the ability to query information from the database and insert tuples, delete tuples and modify tuples in the database, eg., insert, delete, etc.

(iii) Transaction Control - SQL includes commands for specifying the beginning and ending of transactions,

&, commit, rollback, save points, etc.

SQL Clauses

Select Query : (i) For data retrieval in SQL, i/p & o/p both are relations.

(ii) No. of relation i/p to query will be atleast one, but o/p will always be a single relation ~~without~~ without any name unless specified.

(iii) Basic structure of SQL ~~query~~ query consists of three clauses.

(iv) It ~~sho~~ should be noted that ~~it~~ 'select' and 'from' are mandatory and if not required, then it is not necessary to write 'where'.

(v) SQL in general, is ~~not~~ not case sensitive.

(vi) SQL allows duplicates in an i/p relation as well as ~~the~~ in the ~~res~~ result of SQL expression.

Select Clause : (i) It is used to pick column required in result of the query, out of all the columns in the i/p relation.

(ii) Order in which columns are represented in the select clause will be same in the order of columns in the result.

(iii) If all columns are required, then * can be used.

(iv) May also contain arithmetic expression involving operators like +, −, /, *, etc. ~~However~~ However, it does not change the data in the database.

* DISTINCT keyword ~~use~~ removes duplicacy.

Where Clause : (i) It is used to specify conditions or predicates.

(ii) It can have expressions involving comparison operators <, >, <=, >=, <>, etc.

(iii) SQL allows use of logical operators in where clause like AND, OR, NOT, etc.

(iv) It allows 'between' comparison operator to specify where clause. It specifies >= to some value and <= to some other value. Similarly 'not between' is ~~also~~ also present.

Set Operators : (i) SQL provides set operations like Union, Intersect, except/minus/set difference which corresponds to the mathematical set ~~operators~~ operations $\cup$, $\cap$, $-$ respectively.

(ii) Here, union, ~~intersection~~ ~~inter~~ intersect, minus ~~and~~ automatically eliminates duplicate tuples.

(iii) If ~~we~~ we want to retain duplicate tuples, we write 'ALL' after operators.

(iv) Set operations are only applicable between two relations if they are ~~comparable~~ compatible. i.e. having same number of columns with correspondingly same domain.

Cartesian Product : (i) It combines two relations into one by concatenation of each tuple of the first relation with every possible tuple of the second relation.

(ii) From clause by itself defines the C.P. of the relation listed.

(iii) C.P. is commutative in nature, so order doesn't matter.

(iv) If same attribute name appears in both relations, we prefix the name of the relation from ~~which~~ which the attribute originally comes.

Natural Join : (i) It will work same as cartesian product but consider only those pair of tuples with the same value of those attribute that appear in the schema of both relations.

(ii) Notice that common attribute comes only once and prefix is also not required as we allow only some values.

(iii) Commutative in nature.

(iv) May lead to data loss if attribute to be matched have different names, or some values occur only in one table.

Inner Join: (i) It is an operation which works same as cartesian product if used alone.

(ii) But if join is used with keyword 'using', it provides additional functionality.

(ii) Provides ability to explicitly choose columns which must be used by join for comparison & removal of redundant tuples.

(iv) If there are more than one column common between two tables but we don't want each of them to be considered, then we specify which column to be considered.

Join with ON: (i) In SQL, join with 'ON' allows us to use a general predicate over the relations being joined.

(ii) This predicate is written like a 'WHERE' predicate except for the use of the keyword 'ON' instead of 'WHERE'.

(iii) Like C.P., we must specify which attribute must be marked.

(iv) More powerful as predicates of where can also be written with 'ON'.

(v) 'ON' condition can express any SQL predicate.

(vi) It improves the readibility of the query.

(vii) In case of outer join, 'ON' condition behaves differently from 'where'.

Outer Join: One problem with natural join is that only those values

values that appear in both relations will manage to reach the final table. But if some data is explicitly in table one or table two, then it will be lost.

There are 3 forms of outer join:

(i) ~~Let~~ Left outer join (Left join)

(ii) Right outer join (Right join)

(iii) Full outer join (Full join / Complete join)

Rename Operation: (i) SQL Aliases are used to give table or columns of a table, a ~~temp~~ temporary name.

(ii) Aliases only exist for the duration of the query.

(iii) Uses 'AS' ~~clause~~ clause & can appear both in 'select' and ~~from~~ 'from' clauses.

(iv) Often used to name, more readable, meaningful or smaller.

(v) Used for comparing a table with itself.

(vi) New ~~temporary~~ temporary ~~name~~ name of table known as table alias / correlation variable / tuple variable.

(vii) Just creates a new name, but doesn't changes anything in the database.

~~String Operations~~:

Keep practicing SQL questions.

HackerRank and Leetcode