DBMS - 2

31-10-14

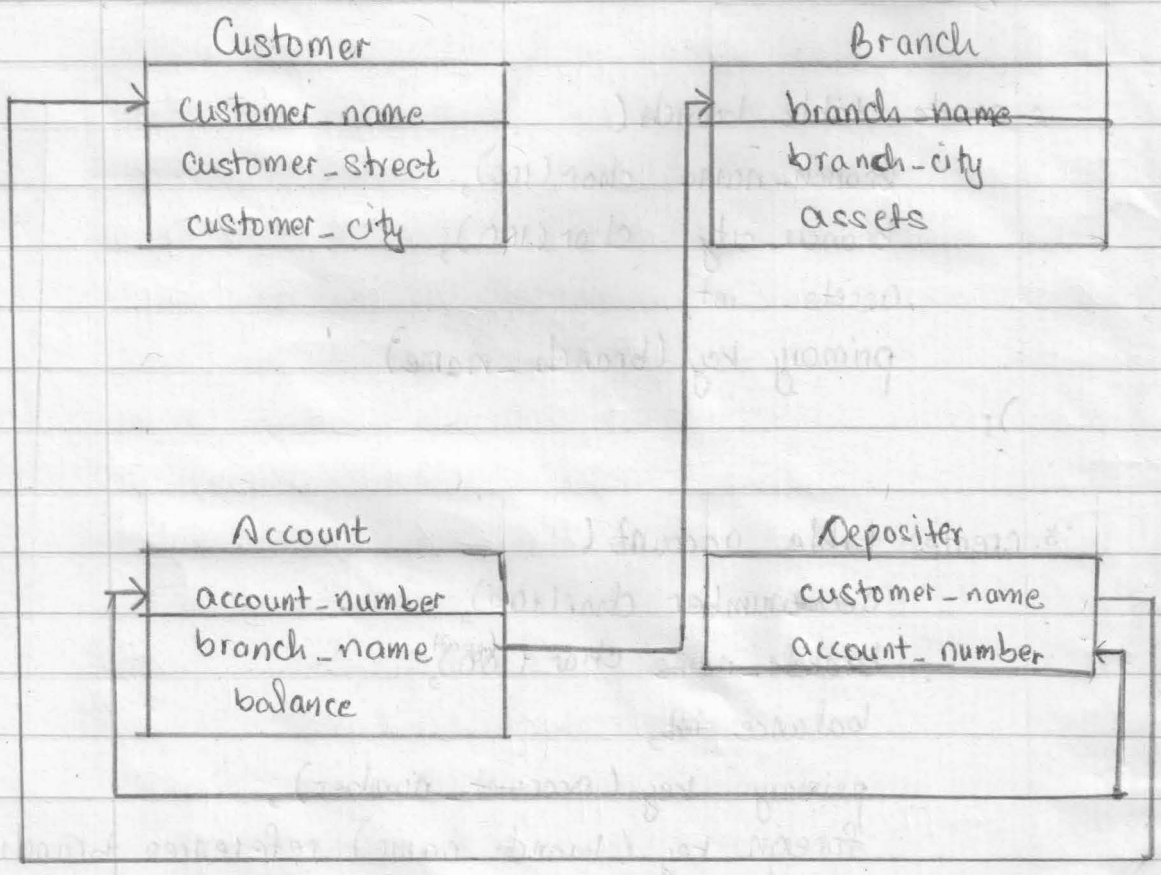
Referential Integrity in SQL

1. Create table customer(
   customer_name char(100),
   customer_street char(100),
   customer_city char(100),
   primary key (customer_name)
);

2. create table branch(
   branch_name char(100),
   branch_city char(100),
   assets int,
   primary key (branch_name)
);

3. create table account(
   account_number char(100),
   branch_name char(100),
   balance int,
   primary key (account_number),
   foreign key (branch_name) references branch
);

4. Create table depositor (
customer_name char(100),
account_number char(100),
primary key (customer_name, account_number),
foreign key (account_number) references account,
foreign key (customer_name) references customer,
);



Assertions

An assertion is a predicate expressing a condition with which the database to always satisfy. Domain constraints, functional dependency and referential integrity are special forms of assertion.

If a constraint cannot be expressed in these forms, we use an assertion.

Example: sum of loan amounts for each branch is less than the sum of all accounts' balances at the branch.

Every loan customer keeps a minimum of thousand dollars in an account.

General syntax for creating assertion in SQL is

create assertion <assertion_name> check <predicate>

1. create assertion sumconstraint check
(not exist (select * from branch
where (select sum (amount) from loan
where loan.branch_name = branch.branch.name)
>= (select sum (amount) from account
whae loan.branch_name = branch.branch name))

Fundamental Dependencies

Functional dependencies are constraints on the set of legal relation. It defines attributes of relation, how they are related to each-other.

It determines unique value for certain set of attributes to the value for another set of attributes i.e. functional dependency is a generalisation of the notation of key.

Fundamental dependencies are interrelationship among attributes of a relation.

"For a given relation R with attributes X and Y, Y is said to be fully functional dependent on X, if given value for each X uniquely determines the value of the attribute in Y. X is called determinant of the functional dependency (F.D) and functional dependency is denoted by $X \rightarrow Y$"

Example: Consider a relation supplier

supplier(supplier_id, sname, status, city)

- supplier.supplier_id → supplier.sname
- supplier.supplier_id → supplier.status
- supplier.supplier_id → supplier.city

} supplier_id ⇒ sname

There are four types of functional dependencies:

i. Full functionality dependency
ii. Partial functionality dependency
iii. Transitive dependency
iv. Trivial functional dependency

10-11-14 Assignment #4

Relational Calculus

- Tuple Relational Calculus
  - Well formed formula
  - Safety of expressions
  - Equivalent expressions
  - Examples of queries in Tuple Relations (Basic operations)
- Domain Relational calculus (Basic operations Only)

11-11-14

Trigger

Trigger is a statement that is automatically executed by the system as a side-effect of a modification to the database. While writing a trigger, we must specify:

- conditions under which the trigger is executed.
- action to be taken when trigger executes.

Triggers are useful mechanism to perform certain task automatically. When certain condition is meet. Sometimes trigger is also called rule or action rule.

Basic Syntax for trigger:

```
Create or Replace Trigger <Trigger_name>
    {Before, After}
        {Insert | Delete | Update [of column, ... ]}
            ON <Table_name>
[Referencing {old as <old>, NEW as <new>} ]
Declare
    variable declaration;
        Begin
        ...
        End;
```

Example:

Suppose that instead of allowing ~~native~~ negative ~~amounts~~ account balances, the bank deals with overdrafts by:

- setting the account balance to zero.
- creating a loan in the amount of the overdraft providing same loan number as a account number of the overdrawn account.

```
create trigger overdraft_trigger
    after update on account
        referencing new row as nrow
            for each row
                when nrow.balance < 0
    begin
        insert into borrower
            (select customer_name, account_number
                from depositor
                    where nrow.account=number,
                    nrow.branch_name, nrow.balance);
        update account
            set balance = 0
                where account.account_number
                = nrow.account_number;
    end;
```

Why triggers can be hard to understand?

In an active database system, when the DBMS is about to execute a statement that modifies the database, it checks if some triggers are activated by the statement. If so, the DBMS process, the trigger by evaluating its condition part, and then (if the condition evaluates to true) executing its action part.

If a statement activates more than one triggers, the DBMS typically processes all of them in one arbitary order. An important point is that the execution of the action part of a trigger could in turn activate another trigger. The execution of the action part of a trigger could again activate another trigger. Such triggers are called recursive triggers. The potential for such chain activations and the unpredictable order in which a DBMS process activated triggers can make it difficult to understand the effect of a collection of triggers.

Constraints v/s Triggers (Do yourself)

Other uses of triggers

i. Trigger can alert users to unusual events (as reflected in updates to the database). For eg: we may want to check whether a customer placing an order has made enough purchase in the last month to qualify for an additional discount. If so, the sales clerk must be informed so that he can tell the customer and possibly generate additional sales.

ii. We can rely this information by using a trigger that checks recent purchase and print a message if the customer qualifies for the discount.

ii. Trigger can generate a log of events to support auditing and security checks

iii. We can use triggers to gather statistics on table accesses and modifications.

13-11-'19

String Operations

like - operator

i. Percent (%) : matches any substrings

ii. Underscore (_) : matches any character

Eg: 'I %' matches any string beginning with I

'ABC%' '% I %'

'%pur' 'I % I'

'% I %' - matches any string containing 'I' as substring

like '___' matches any string of exactly three characters.

like '___%' matches any string of ~~exactly~~ atleast three characters

List all customers whose name starts with S.

```
select customer_name from customer
    where customer_name like 'S%';
```

24-11-14

Assignment # 5

1. Trivial and non-trivial full dependencies, closure of a set of functional dependencies. Attribute closure FDs, irreducible set of FDs
2. Transitivity, reflexivity and augmentation properties of FDs.

1-12-14 Lab assignment # 7

Aggregate functions

min, max, sum, avg, count, count*

1. How many tuples are stored in the relation employee.
2. How many different job titles are stored in the job title employee?
3. List the minimum and maximum salary.
4. Sum of all salaries of employees working in dept 30.

2-12-'14

Anomalies

An anomaly is simply an error or inconsistency in the database. A poorly designed database runs the risk of introducing numerous anomalies. There are three types of anomalies:

i. Insertion anomalies

An anomaly that occurs during the insertion of a record. It is the inability to add data to the database due to absence of other data.

ii. Deletion anomalies

An anomaly that occurs during the deletion of a record. It is the unintended loss of data due to deletion of other data.

iii. Update anomalies

An anomaly that occurs during the updating of a record. For example, updating a description's column for a single part in an inventory database require you to make a change to thousands of rows.

Pitfall of relational model:

The main goal of relational database is:

i. to create / find good collection of relational schemas such that database should allow us to store data / information without unnecessary redundancy and should also allow to retrieve information easily, i.e. the goal of relational database design should be concentrated
   - to avoid redundant data from database
   - to ensure that relationship among attributes are represented.
   - to facilitate the checking of updates for the violation of data integrity constraints

ii. A bad relational database design may lead to
   - repetition of information i.e. it leads data redundancy in database, so obviously it requires much space
   - inability to represent certain information

   Example: Consider a relational schema
   branch_loan = (branch_name, branch_city, assets, customer_name, loan_no, amount)

branch_name	branch_city	assets	customer_name	loan_number	amount
Kathmandu	Baneshwor	25000	Rohan	L-15	3000
Lalitpur	Patan	19000	Mohan	L-17	5000
Kathmandu	Baneshwor	25000	Raju	L-19	10000
Pokhara	Prithvinagar	17000	Manoj	L-10	7000
Lalitpur	Patan	19000	Swikar	L-30	9000

Redundancy

i. Data for branch_name, branch_city, assets are repeated for each loan that is provided by bank

ii. Storing information several times leads wastage of storage space / time.

iii. Data redundancy leads problem in insertion, deletion and update

i Insertion Problem We cannot store information about a branch without loan information, we can use NULL values for loan information but they are difficult to handle.

ii Deletion problem

In this example, if we delete the information of Manoj (i.e. delete from branch_loan where

customer_name = 'Manoj'), we cannot obtain the information of Pokhara branch (i.e. we don't know branch_city of Pokhara, total assets of Pokhara branch, etc.)

iii. Update Problem

Since data are duplicated, so multiple copies of same fact need to update while updating one. It increases the possibility of data inconsistency when we made update in one copy, there is possibility that only some of the multiple copies are updated but not all, which lead data / database in inconsistent state.

Decomposition

The idea of decomposition is breakdown of large and complicated relation in number of simple and small relations which minimized data redundancy. It can be considered principle to solve the relational model problem.

Lossless join decomposition - Do yourself

4-12-'14

10-12 marks

NORMALIZATION

Concept of normalization

Relational database is a collection of tables. The database must be carefully designed in order to get full advantage. By designing database tables carefully, we can save space, minimize duplicate data, protect data consistency as well as provide faster transaction by sending less data.

Normalization is the answer to achieve these benefits. It is a process of splitting tables to reduce data redundancy (duplicate data) and establishing relation between tables. Normalization provide flexibility, data consistency (or data integrity) and avoid anomalies while inserting, deleting, updating data.

The following figure illustrates the unnormalized and normalized database.

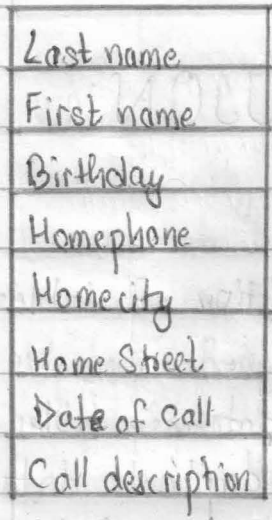


Fig Unnormalized form

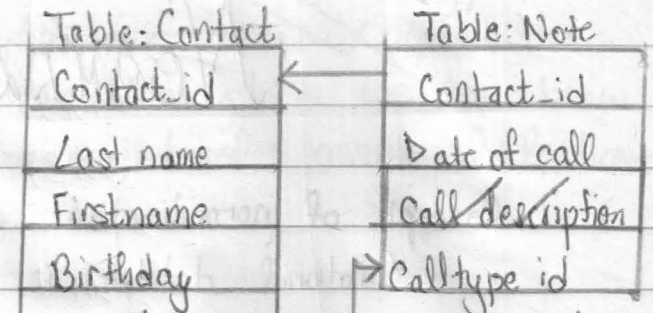


Fig Normalized form

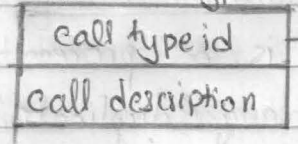


Fig. Next stage of normalized table

Needs of Normalization - Do yourself.

Objectives of Normalization

- to free a collection of relation from undesirable insertion, update and deletion.
- to move the relational model more informative to use.
- to increase the lifetime of application programs.

Properties of ~~Normalization~~ Relation after normalization

- no data value should be duplicate in different row or tuple.
- a value must be satisfied for every attribute in a tuple.
- important information should not be accidentally lost
- when new data/record are inserted to relation, other relation in a database should not be affected.

Forms of normalization

The breaking down of a table may undergo series of stages called as normal form or just an NF in short. There are 6 normal forms so far, which are widely used. The higher level of normalization can not be achieved unless previous levels have been satisfied. For example, to be in second normal form, it should already be in first normal form.

- First normal form (1NF)
- Second normal form (2NF)
- Third normal form (3NF)
- Boyce Codd Normal form (BCNF)
- Fourth normal form (4NF)
- Fifth normal form (5NF)

First Normal Form (1NF)

A table is said to be first normal form if it has no repeating groups i.e. for each cell in a table, there can be only one value. If a group of ~~five~~ items repeat, it should be split into a new table.

Let us consider on example of billing system of a store.

Sales_order no
Date
Customer No
Customer Name
Customer Address
Item No
Description
Quantity
Unit price

Fig. unnormalized table

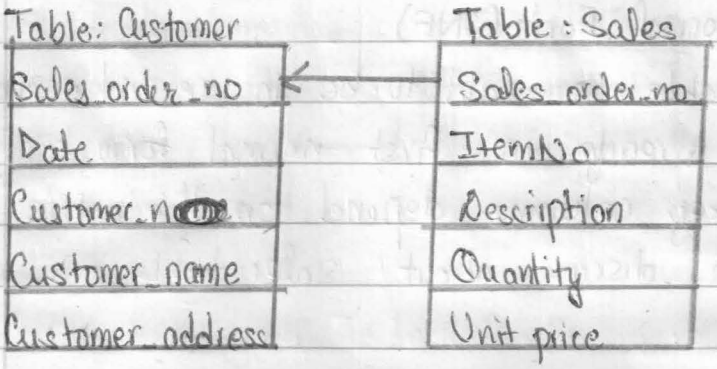


Fig. Table in 1NF

The fields date, customer_no, customer_name, customer_address are repeating fields and the fields itemno, description, quantity, unit price are non-repeating fields. If all fields are kept in a single table, header data are repeated for every line in sales order i.e. to keep record for each items' information on customer should also be entered each time.

In 1NF, repeating groups are separated into new table as shown in figure above.

These 2 tables can be linked together by a common field - sales_order_no. Now, these 2 tables are in 1NF

Second Normal Form (2NF)

A table is said to be in second normal form if it is already in first normal form and if every non-key columns depend on the entire key.

Lets discuss about sales table.

Table: Sales

Sales_order_no
ItemNo
Description
Quantity
Unit price

Here, the key field is sales_order_no. The field description depends on ItemNo rather than sales_order_no. However, quantity and unit price are not dependent on item no as they may be different for different sales order. In this case, the table is split again. The column that depend on the key are kept in one table and rest on another table as follows:

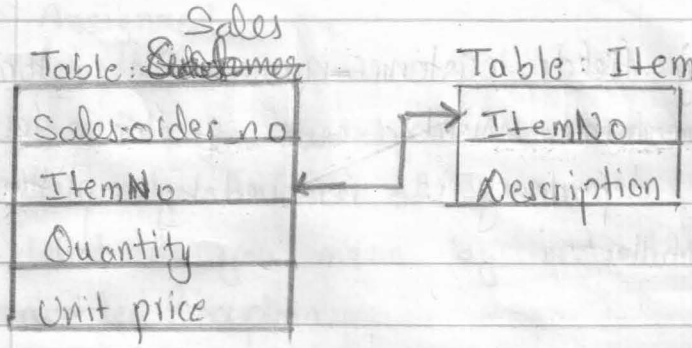
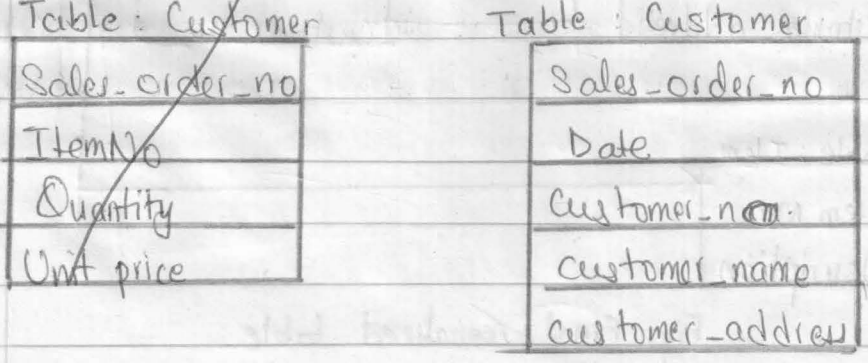


Fig Table in 2NF

Now, these two tables are in 2NF and they can be linked together by common field ItemNo.

Third Normal Form (3NF)

A table is said to be in 3NF if it is already in 2NF and if non-key columns are not dependent on each-other. There should not be any hidden dependencies among non-key columns. Lets discuss about customer table.



Here, non-key fields customer_name and customer_addr depends on customer number but not on sales_order_no. This hidden dependency is removed by splitting the table as follows:

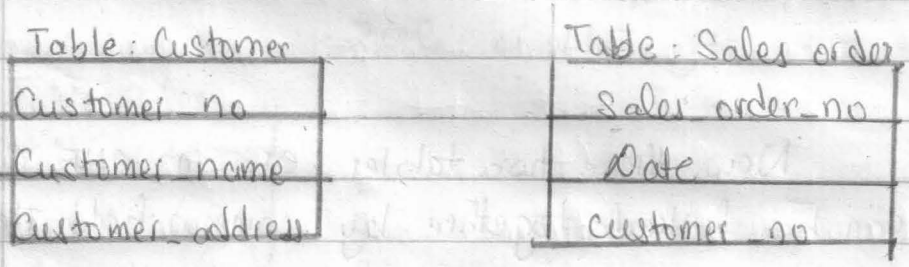


Fig. Table in 3NF

Now, the final normalized tables after third normal form will be as follows:

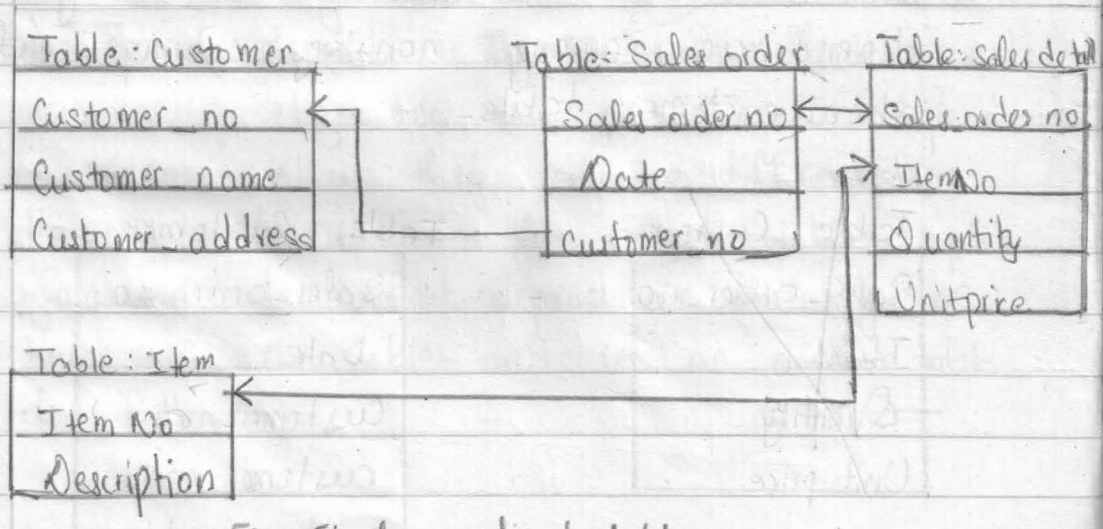


Fig. Final normalized table

8-12-14 Assignment

~~Lab~~ # 7

1. What do you mean by relational database design? List out the characteristics of RDBMS.
2. What do you mean by redundancy? How can this be avoided? Explain.
3. Define term anomolies. Explain BCNF in detail.
4. Distinguish the differences between 2nd and 3rd NF.
5. What do you mean by functional dependency. What role does it play in normalization? Explain.
6. What is transitive dependency? How does it differ from full functional dependency? Explain.
7. Why is concurrency control needed? Explain lost update, inconsistent retrieval and uncommitted dependency anomalies.

3-12-'14

TRANSACTION MANAGEMENT

Recovery

Recovery in a database system means, primarily, recovering the database itself, i.e. restoring the database to a state that is known to be correct (or, rather, consistent) after some failure have rendered the current state inconsistent, or atleast suspect.

Commit

The commit operation signals successful end of transaction, it tells the transaction manager that a logical unit of work has been successfully completed, the database is in a consistent state again and all of the updates made by that unit of work can now be "committed" or made permanent.

Rollback :

The rollback operation signals unsuccessful end of transaction, it tells the transaction manager that something has gone wrong, the database might be in an inconsistent state, and all of the updates made by the logical unit of work so far must be "rolled back" or undone.

Transaction

Transaction begins with successful execution of a begin transaction. statement and it ends with successful execution of either a commit or a rollback statement.

A commit point thus corresponds to the end of a logical unit of work and hence to a point at which the database is or should be in a consistent state. Rollback, by contrast, rolls the database back to the state it was in at begin transaction, which effectively means back to the previous commit point.

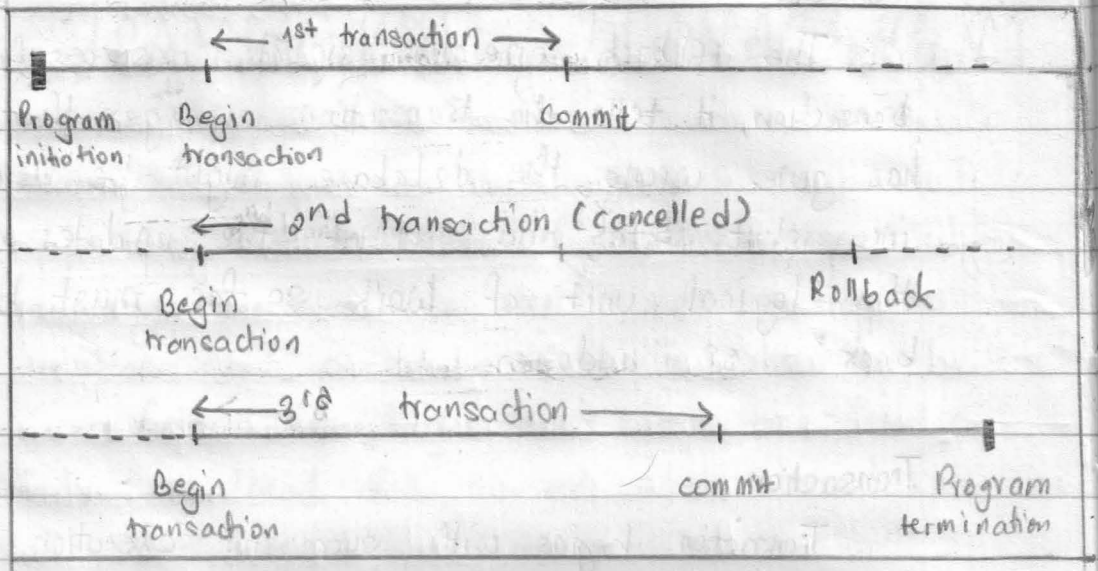


Fig. Program collection is a sequence of transactions

When a commit point is established

- all updates made by the executing program, since the previous commit points are committed, i.e. they are made permanent. Once committed, an update is guaranteed, never to be undone
- all database positioning is lost and all tuple locks are released. Database positioning here refers to the idea that at any given time, an executing program will typically have addressability to certain tuples

The ACID properties

- Atomicity — All or nothing
- Consistency
- Isolation
- Durability

- Transactions are **atomic** (all or nothing)
- Transactions preserve database **consistency**, i.e. a transaction transforms a consistent state of a database into another consistent state, without necessarily preserving consistency at all intermediate point

11-12-14

System Recovery

The system mostly prepared to recover, not only from purely local failures such as the occurrance of an overflow condition within an individual transaction but also from global failures such as power outage.

A local failure, affects only the transaction in which the failure has actually occurred, ~~such failure ha~~

A global failure, affects all of the transaction in progress at the time of the failure, and hence the significant system ~~wide implementation~~ wide implication.

Failures fall into two broad categories:

i. System failures (E.g. power outage), which affect all transactions currently in progress but do not physically damage the ~~syste~~ database. A system failure is sometimes called ~~soft~~ crash.

ii. Media failures (E.g. head crash on the disk), which do cause damage to the database, or to some portion of it, and affect atleast those transaction currently using that portion. A media failure is sometimes called a hard crash.

Media Recovery

A media failure is a failure such as a disk head crash or a disk-controller failure in which some portion of the database has been physically destroyed. Recovery from such a failure basically involves reloading (or restoring) the database from a backup copied and then using the log both active and archived portion.

The need to be able to perform media recovery ~~comprise~~ implies the need for a dump/restore (or unload/reload) utility.

After a media failure, the restore portion of the ~~utility~~ is used to create the database from a ~~Utility~~ specified backup copy.

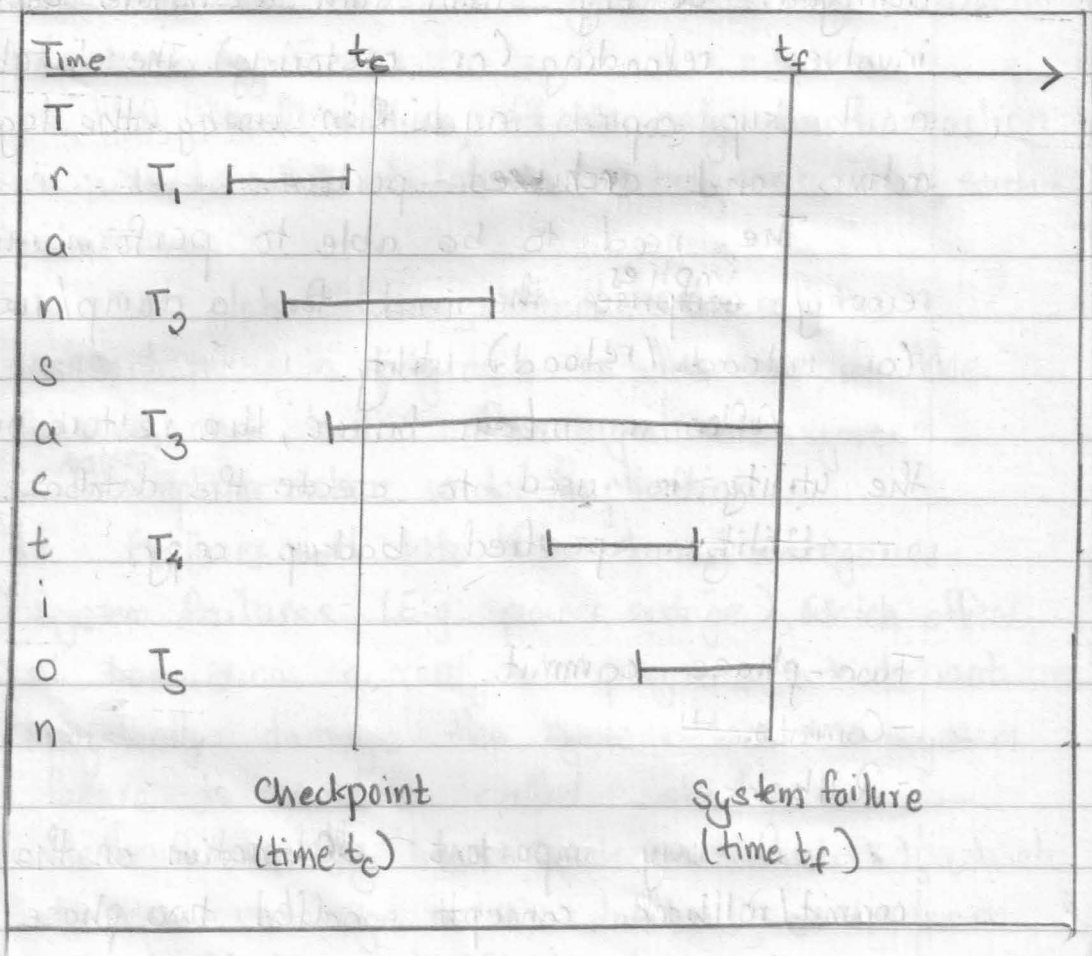
Two-phase commit.

- Commitment
- Rollback

A very important elaboration on the basic commit/rollback concept called two phase commit. Two-phase commit is important whenever a given transaction can interact with several independent resource manager, each managing its own set of

recoverable resources and maintaining its own recovery log

Transaction categories



5 transaction categories

- a system failure has occurred at time t_f.
- the most recent checkpoint prior to time t_f was taken at time t_c.
- transaction type T_1 completed (successfully) prior to time t_c.
- transaction type T_2 start prior to time t_c and completed after time t_c and before time t_f.
- transaction type T_3 also started prior to time t_c but did not complete by time t_f.
- transaction type T_4 started after time t_c and completed before time t_f.
- finally, transaction type T_5 also started after time t_c but did not complete by time t_f.

Concurrency

The term concurrency refers to the ~~cracke~~ fact that DBMSs typically allow many transactions to access the same database at the same time, and in such a system as is well-known, some kind of concurrency control mechanism is needed to ensure that concurrent transactions do not interfere with each other.

Three Concurrency Problems

i. the lost update problem

ii. the uncommitted dependency problem

iii. the inconsistent analysis problem

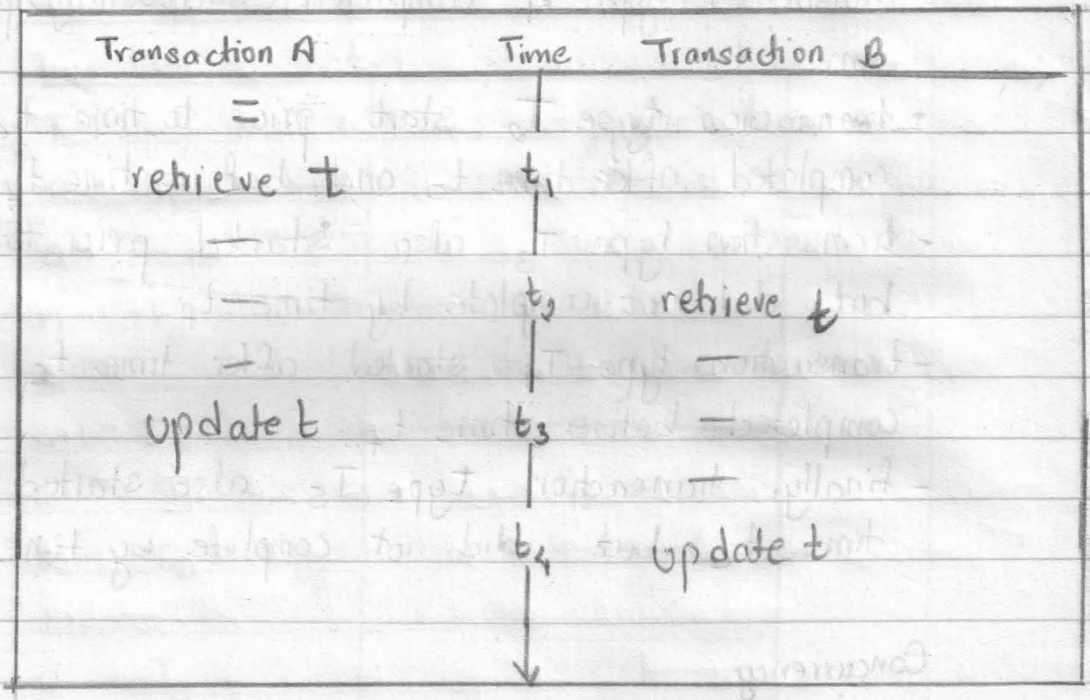


Fig. Transaction A lose an update at time t

Transaction A retrieves some tuple t at time t_1; transaction B retrieves that same tuple t at time t_2.

Transaction A updates the tuple (on the basis of the values seem at time t_1) ~~At time t_3~~ and

transaction B updates the same tuple at t_4.

Transaction A's update is lost at time t_4, because transaction B overwrites it without even looking at it.

Lab # 8

Database recovery

- Purpose of database recovery
- Types of failure
- Storage hierarchy
- Buffer management
- transaction log

16-12-14

The uncommitted Dependency Problem

The uncommitted dependency problem arises if one transaction is allowed to retrieve, or worse, update - a tuple that has been updated by another transaction but not yet committed by that other transaction. For if it has not yet been committed, there is always a possibility that it never will be committed but will be rolled back instead - in which case the first transaction will have seen some data that now no longer exist (and in a sense never did exist)

In the first example (which is shown below), transaction A sees an uncommitted update (also called an uncommitted change) at time t_2. That update is then undone at time t_3.

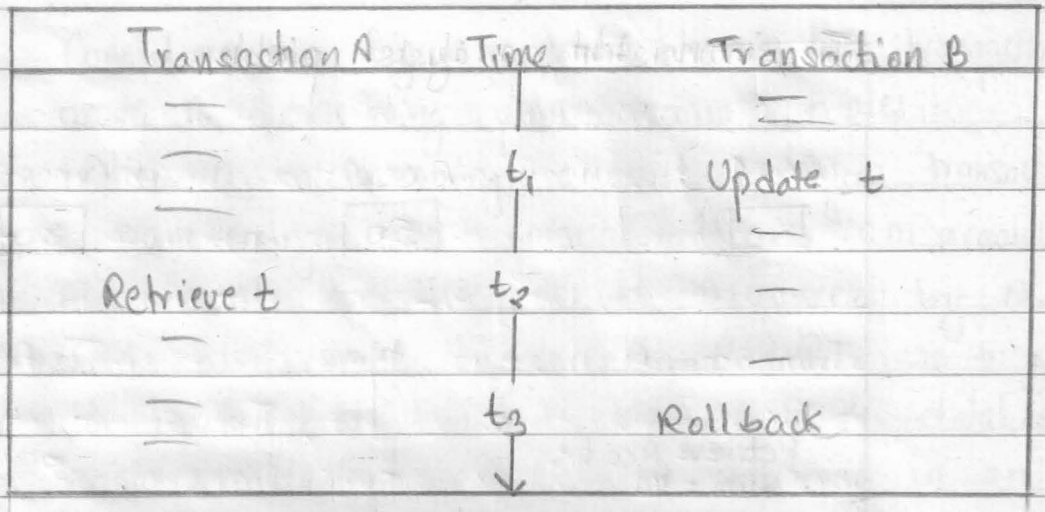


Fig a Transaction A becomes dependent on an uncommitted change at time t_2

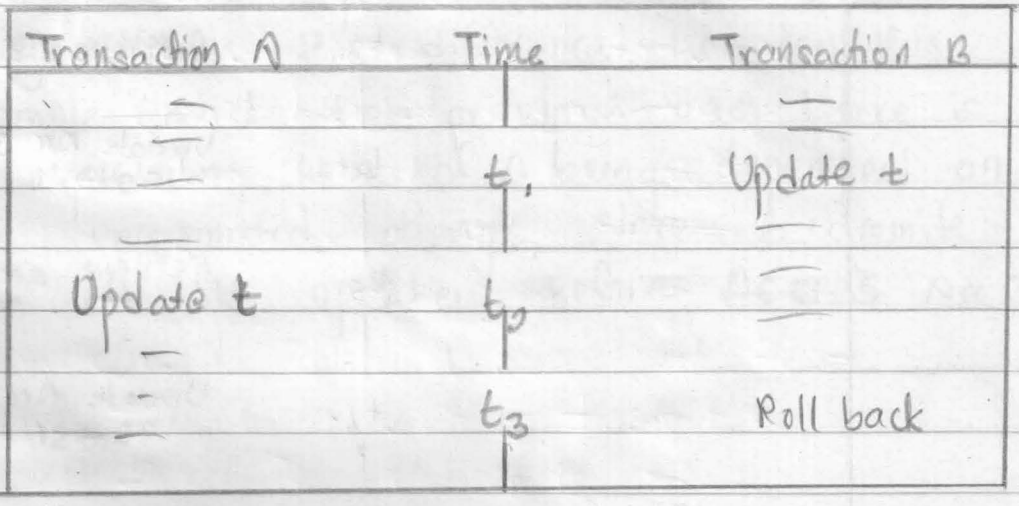


Fig. b Transaction A updates an uncommitted change at time t_2, and loses that update at time t_3.

The inconsistent analysis problem

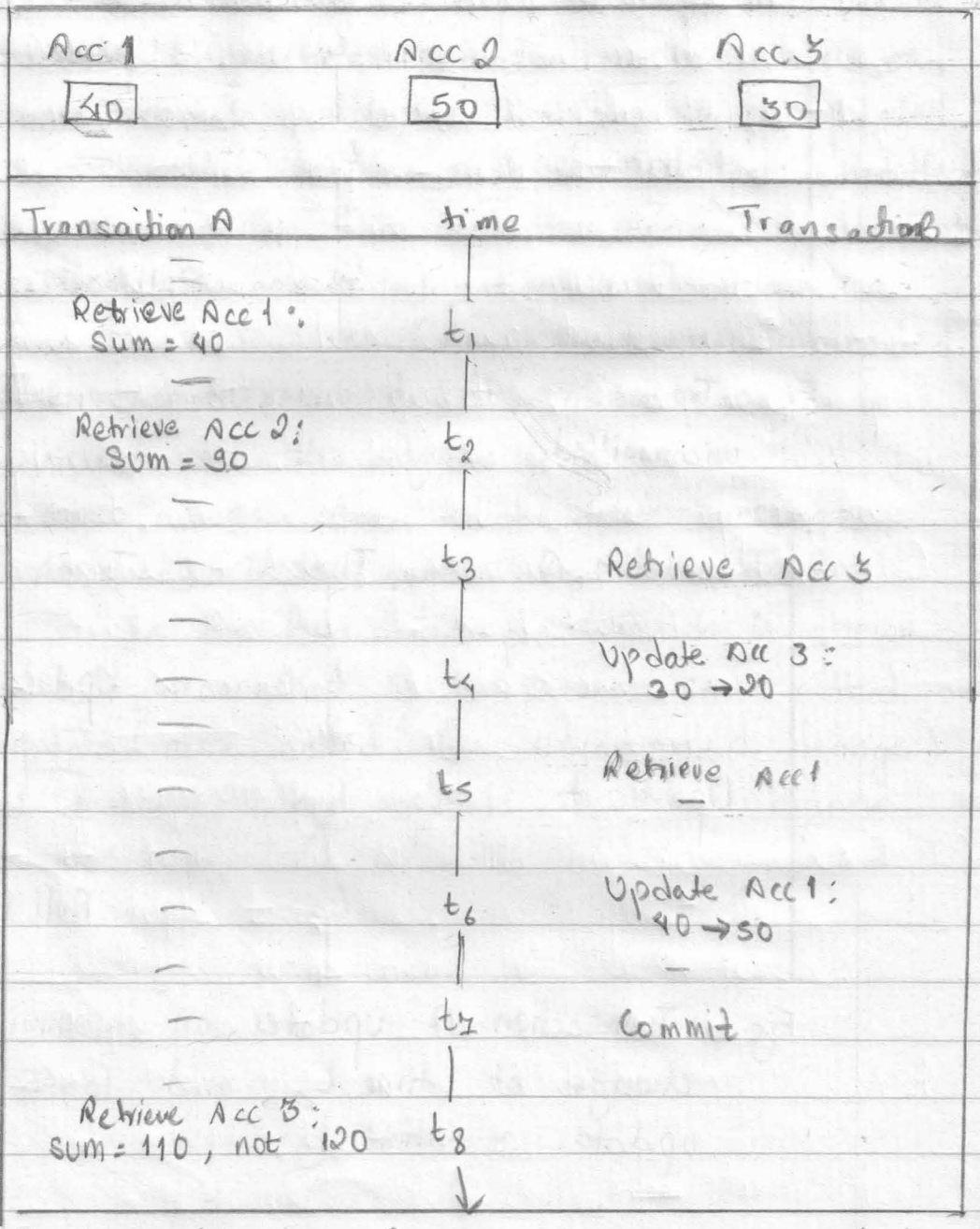


Fig. Transaction A performs an inconsistent analysis

Consider the figure, which shows two transactions A and B operating on account tuples: transaction A is summing account balances, transaction B is transferring an amount 10 from account 3 to account 1. The result produced by A, 110 is obviously incorrect; if A were to go on to write that result back into the database, it would actually leave the database in an inconsistent state. We say that A has seen an inconsistent state of the database and has therefore performed an inconsistent analysis. Note the difference between this example and the previous one: there is no question here of A being dependent on an uncommitted change, since B commits all of its updates before ~~Access~~ Acc 3

~~Locking~~ Locking

A concurrency control technique is called locking. When a transaction needs an assurance that some object it is interested in - typically a database tuple - will not change in some manner while its back is turned, it acquires a lock on that object. The effect of the lock is to "lock other transactions out of" the object in question, and thus, in particular prevent them from changing it.

- first, we assume the system supports two kinds of locks, exclusive locks (X-locks) and shared locks (S-locks)

Note: X and S locks are sometimes called ~~write~~ write locks and read locks respectively.

- if transaction A holds an exclusive (X-lock) on tuple t, then a request from some distinct transaction B for a lock on either type on t will be denied.
- if transaction A holds on a shared (S-lock) on tuple t, then
  - A requests from some distinct transaction B for an X-lock on t will be denied.

- A request from some distinct transaction B for an S-lock on t will be granted (i.e. B will now also hold on an S-lock on t).

Deadlock

Deadlock is a situation in which two or more transactions are in a simultaneous ~~weight~~ wait state, each of them waiting for one of the others to release a lock before it can proceed. Locking can introduce problems of its own, principally the problem of deadlock.

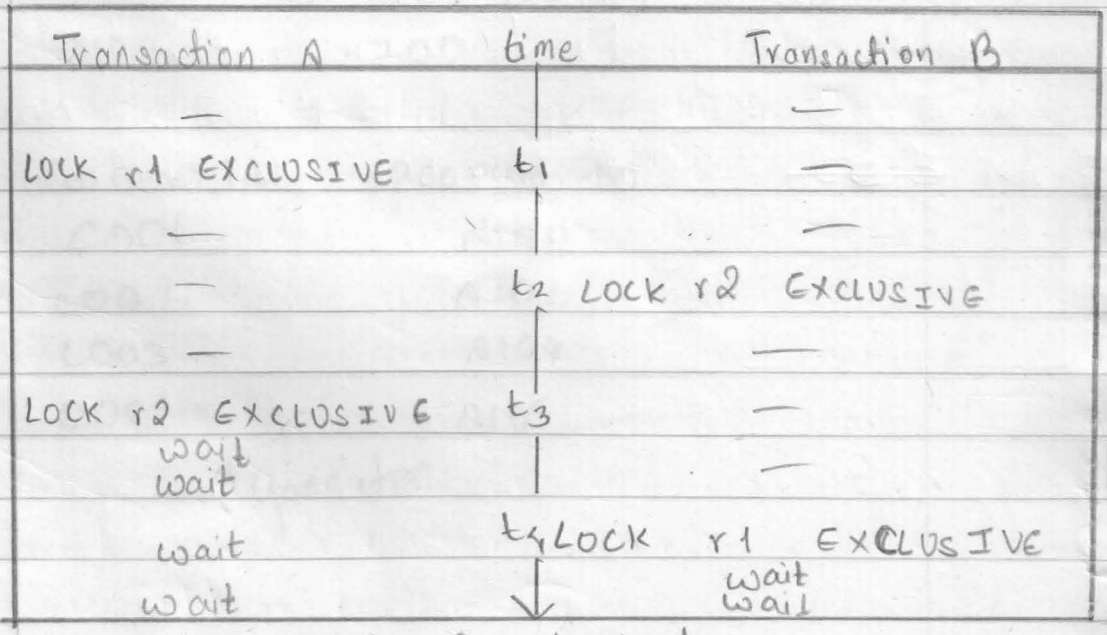


Fig. An example of deadlock

~~A deadl~~ The figure shows a deadlock involving two transactions, but deadlocks involving three, four, ... transactions are also possible, atleast in principle; we remark, however, that experiments with system R seemed to show that in practice, Deadlocks almost never do involve more than two transactions.

18-12-14

Lab

Customer_id	Customer_name	Address
C001	Smith	Ktm
C002	John	Bhaktapur
C003	Semi	Lalitpur
C004	Ivan	Ktm

Customer

account_no	balance
A101	900
A102	300
A103	200
A104	700

account

Customer_id	account_no
C001	A101
C002	A102
C003	A104
C004	A105

depositor

1. Find the name of customer whose customer Id is C001
2. Find the name and balance of the customer
3. Insert record to the customer table, customer_id: C005, customer_name: Michael, address: ktm
4. Delete record from depositor whose customer_id is C004.
5. Increase the balance by 5% in account table whose account no is A101 or current balance is only 200.
6. Find all customer names
7. Find all the different customes address.
8. Find the customer_id and its corresponding customer_name.
9. List name and address of customer who stay in Kathmandu.
10. List all customer whose name is Smith and address should be Ktm
11. List the customer_id, acc_id & balance whose balance is more than 300.

Select customer.customer_id, account.account_no, account.balance from customer, account, depositor where ((customer.