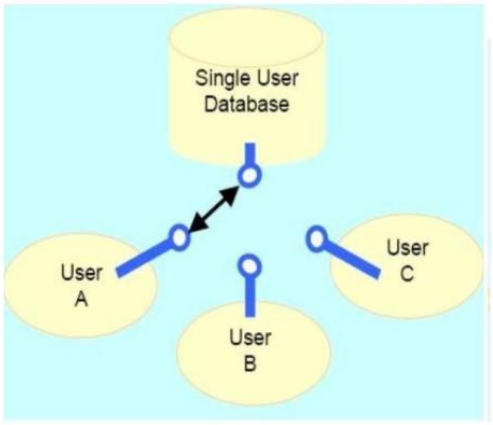
Database 2

❖ **TRANSACTION PROCESSING**

SINGLE-USER vs MULTI-USER SYSTEMS:

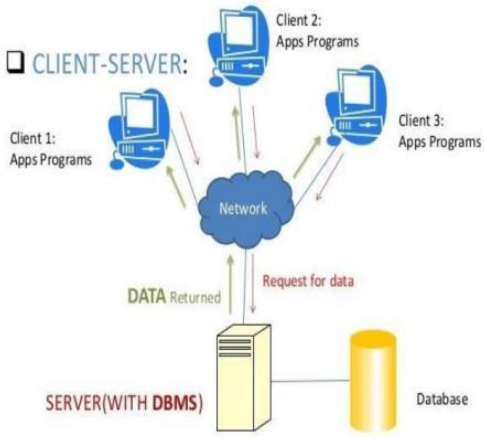


• **Single user system:**

- It is designed to be used by only one user at a time.
- It is typically installed on a single computer and can only be accessed by the user who installed it or the user who is currently logged in.

• **Multi user system:**

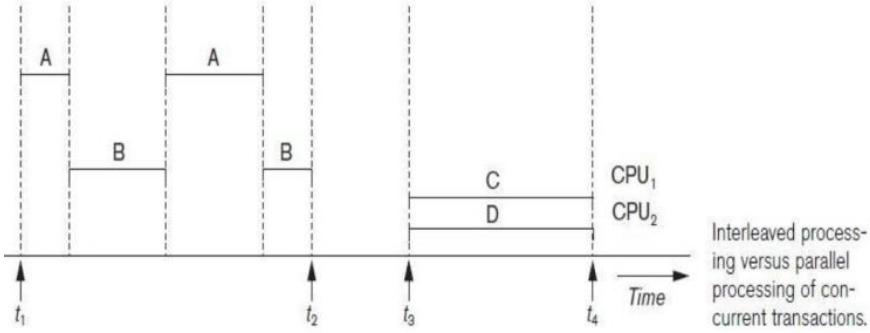
- It can be accessed by multiple users simultaneously.
- It is typically installed on a network server and can be accessed by users who are logged in to the network.



INTERLEAVED vs PARALLEL PROCESSING:

• **Interleaved processing:** It is a process in which more than one task is being processed at the same time.

• **Parallel processing**: It is a process in which the tasks are divided into small sub-tasks that are processing simultaneously or parallel



Interleaved processing versus parallel processing of concurrent transactions.

A transaction :

- it is the logical unit of database processing that includes one or more database access operations (Read: retrieval or Write: insertion, deletion, modification).
- It is the atomic unit of work that is either completed in its entirety or not done at all.
- In the application program it may contain several transactions separated by transaction boundaries.

IMPORTANCE OF CONCURRENCY CONTROL

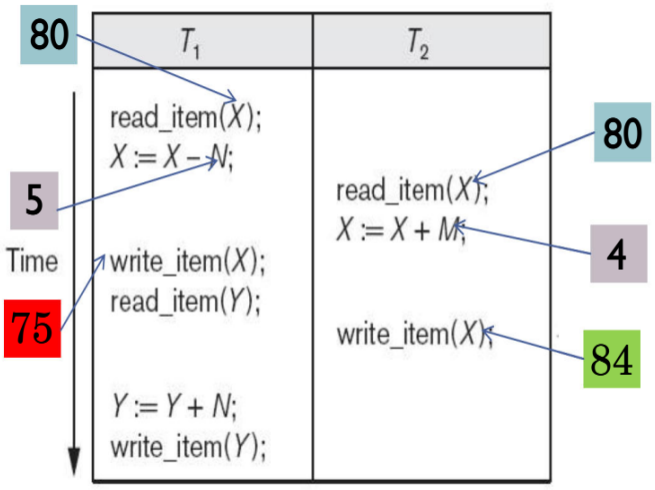
•The lost update.

•The temporary update (Dirty read).

•The incorrect summary problem.

•The unrepeatable read problem.

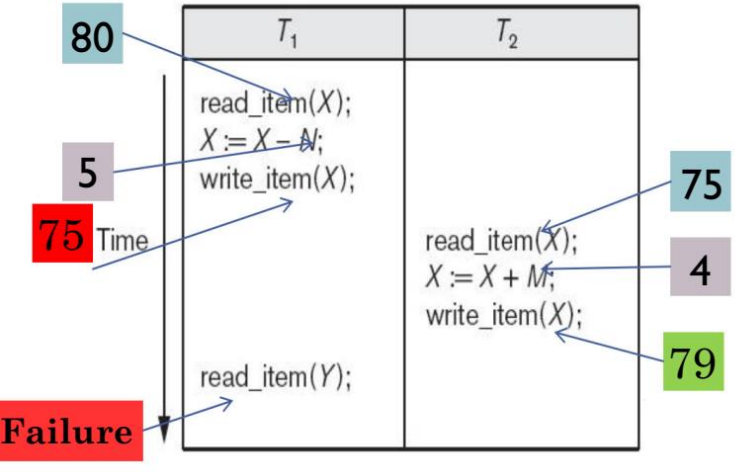
Lost update problem: This problem occurs when **two transactions** that access the same database items have their operations **interleaved** in a way that makes the value of some database items incorrect.



اول مشكله عندنا هى ال *lost update* و دى بيقولك لما يكون ف اتنين *transactions* و الاتنين بيستخدمو نفس الداتا بيز بحيث ان عملياتهم *interleaved* يعنى بيحصلو مع بعض ف نفس الوقت و مترتبين ع بعض و ركز ع الى علمتلك عليه ده عشان هنحتاجه تحت ف المثال ف بيقولك ده ممكن يخلى ان قيم بعض العناصر الى ف الداتا بيز تطلع غلط .

مثال هيفهمك الدنيا شوية .. ف المثال الى ع اليمين ده عندك داتا بيز فيها اتنين *transactions* الى هم *T1, T2* و *T1* هيشتغل الاول هيقرأ قيمه المتغير *X* و بعدين يعمل عليها عمليه و قبل م يسجل قيمه ال *X* بعد م اتغيرت راح قرأ قيمتها القديمه و عمل عليها عمليه تانيه ف ده نتج عنه انه لما خزن البيانات طلع قيمتين للاكس مختلفين ع بعض

Temporary update (Dirty Read): This problem occurs when **one transaction updates a database** item and then the **transaction fails** for some reason "هنعرفهم بعدين اتقل"Meanwhile, **the updated item is accessed (read)** by another transaction before it is changed back to its original value.



تانى مشكله عندنا هى ال *temporary update* و دى بتتسمى بال *Dirty Read* بيقولك ان دى بتحصل لما *transaction* يعمل ابديت لعنصر و بعدين ال *transaction* ده يفشل او يكراش لسبب معين "الى زى م قولتلك فوق هنتكلم عنهم بعدين" المهم ان ف نفس الوقت العنصر الى اتعمله ابديت ده *transaction* تانى استخدمه بعدها علطول قبل م يرجع لقيمته الاساسيه . مثال هيفهمك شويه الدنيا .. ف المثال الى ع اليمين ده عندك داتا بيز فيها اتنين *transactions* الى هم *T1, T2* و *T1* هيشتغل الاول هيقرأ قيمه المتغير *X* و بعدين يعمل عليها عمليه و يسجله علطول (و ده الى محصلش ف اول مشكله) و بعدين *transaction* تانى هيستخدم قيمه ال *X* الجديده الى اتسجلت خلاص و يعمل عليها عمليه تانيه و يسجلها و بعدين حصل ف ال *transaction* الاول *Failure* و هو بيحاول يقرأ ال *X* ف ده هيوقف ال *transaction* ده كله بس ف نفس الوقت زى م قولنا خلاص ف *transaction* تانى استخدم العنصر الى اتعمله ابديت

Incorrect summary problem: This problem occurs if **one transaction is calculating an aggregate summary function** on a number of database items **while other transactions are updating some of these items**, the aggregate function may calculate some values before they are updated and others after they are updated

T_1	T_3

	sum := 0;
	read_item(A);
	sum := sum + A;
	. . .

read_item(X);
X := X − N;
write_item(X);	

	read_item(X);
	sum := sum + X;
	read_item(Y);
	sum := sum + Y;

read_item(Y);
Y := Y + N;
write_item(Y);	

تالت مشكله و دى ابسط نوع فيهم هى ال *incorrect summary* ف المشكله دى بكل بساطه بتحصل لما يكون عندك اتنين *transactions* واحد منهم فيه داله ال *sum* بتحسب مجموع عنصر معين و ف نفس الوقت العنصر ده بيتعمل لقيمته ابديت ف *transaction* تانى ف ده بيخليك يبقى ف قيمتين لداله ال *sum* واحده قبل الابديت و واحده بعد الابديت . مثال .. ف المثال الى ع اليمين ده عندك داتا بيز فيها اتنين *transactions* الى هم *T1, T2* و *T2* هيعمل عمليه ال *sum* للمتغير *X* الاول قبل م يحصل ابديت لاى عنصر بعدها علطول ف ال *transaction* الاول هيحصل ابديت لقيمه ال *X* ف لما هيرجع بعدها لل *transaction* التانى عشان يعمل عمليه *sum* تانيه هيبقى عنده قيمه جديده مختلفه لل *X* بعد الابديت ف ده الى بيقولك عليه هيبقى ف قيمتين مختلفين لل *sum* نتيجه ان ال *X* هيبقى ليها هى كمان قيمتين مختلفين واحده بعد الابديت و التانيه قبل

Unrepeatable read problem: This problem occurs when a transaction T **reads the same item twice** and **the item is changed by another transaction T between the two reads**. Hence, T receives different values for its two reads of the same item.

اخر مشكله عندك بيكلمك ع ال *unrepeatable read problem* و الى هى بتحصل لما يكون ف نفس ال *transaction* بيقرأ قيمه ال *X* مرتين بس بين المرتين دول ف *transaction* تانى عمل ابديت لقيمه ال *X* و متنساش احنا قولنا ان كل دول متوصلين مع بعض *interleaved* و انت فاهم بقى خلاص يعنى ايه .. ف لما هيحصل ابديت بين العمليتين هيخلى ف نفس ال *transaction* يبقى ف قيمتين لل *X* مختلفين ع

IMPORTANCE OF RECOVERY

- It means that the system is responsible for making sure that:
  - All the operations in the transaction are **completed successfully** and their **effect** is recorded permanently in the database.
  - The transaction **does not have any effect** on the **database** or **any other transactions**. (means that there is no transaction with failure that will effect the database or any other transaction)

REASONS FOR TRANSACTION FAILURE:

• Computer failure: If **the hardware crashes**, the contents of the computer's internal memory may be lost

•بيحصل لما ال ***hardware*** يكراش او مثلا ان محتوى الذاكره الداخليه اتمسح

• Transaction or system error: **Transaction operation error** (integer overflow or division by zero). **Erroneous parameter values** or a logical programming error. The **user** may interrupt the transaction during its execution

• بيحصل لما مشكله ف العمليات الى بتتعمل جوا ال ***transaction*** زى مثلا انه يكتب رقم غلط او يقسم ع صفر

او لما تدخل قيمه متغير غلط او يبقى ف ***logical programming error***

او لما اليوزر نفسه هو الى يوقف ال ***transaction*** و هو بيتنفذ

• Local errors: Data for the transaction may not be found.
A programmed abort in the **transaction** causes it to fail

• بيحصل لما مثلا الداتا بتاعت ال ***transaction*** الجهاز او البرنامج مش عارف يلاقيها

او لما يحصل ***abort*** جوا ال ***transaction*** نفسه ف يخليه يحصله ***fail***.

• Concurrency control enforcement: The **concurrency control method** may decide to abort the transaction, to be **restarted later**.

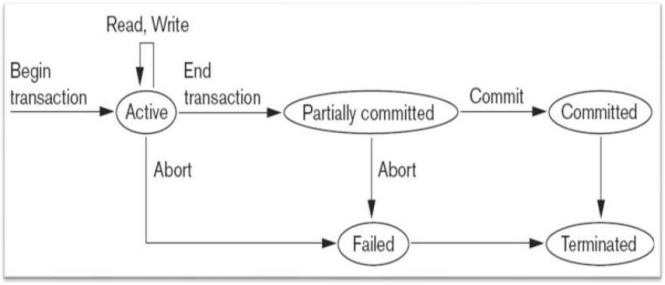
• بيحصل لما ال ***concurrency control method*** هى الى توقف ال ***transaction*** عشان تعمله ريستارت

• Disk failure: disk read/write head crash.
• Physical problems and catastrophes: Ex. power or air-conditioning failure, fire, theft, sabotage, or overwriting disks.

TRANSACTION AND SYSTEM CONCEPTS

- A transaction is the **atomic unit** of **work** that is either completed in its entirety or not done at all.
- For **recovery** purposes, the system needs to keep track of when the transaction **starts**, **terminates**, and **commits** or **aborts**.

TRANSACTION STATES:



- Active state
- Terminated state.
- Partially committed state.
- Committed state.
- Failed state.

Recovery manager keeps track of the following operations:

- **Begin-transaction:** This marks the **beginning** of transaction execution.

- **Read or Write:** These specify **read** or **write** operations on the database items that are executed as part of a transaction.

End-transaction: This specifies that **read** and **write** transaction operations **have ended** and **marks the end limit** of transaction execution.

- At this point it may be necessary to check whether **the changes introduced by the transaction** can be permanently **applied to the database** or whether **the transaction** has to be **aborted** because it violates concurrency control or for some other reason.

بيقولك في ال *end-transaction* **بيحدد امتى ال** *read and write transaction* **خلصت**
و بيحدد كمان اخر transaction execution

- **و كمان ف المرحله دى بيبقى لازم تحدد اذا كان التغيرات الى عملتها ال** *transaction* **اتنفذت بشكل صحيح و هتقدر تطبق ع الداتا بيز او ان ال** *transaction* **لازم يتعمله** *abort* **عشان فيه اى مشكله م الى اتكلمنا عنهم ف ال** *concurrency control* **او لاى سبب تانى**

Commit-transaction: This signals a **successful end** of the transaction so that any changes (updates) executed by the transaction can be safely committed to the database and **will not** be undone.

- **Rollback (Abort):** This signals that the transaction **has ended unsuccessfully**, so that any changes or effects that the transaction may have applied to the database **must be** undone.

SYSTEM LOG:

- It keeps track of **all transaction operations** that affect **the values of database items**.
- It may be needed to permit **recovery** from **transaction failures**.
- The log is kept on disk, so it **is not affected** by any type of failure **except** for **disk or catastrophic failure**.
- The log is periodically **backed up to archival storage (tape)** to guard against such **catastrophic failures**.

Commit point of a transaction:

- A transaction T reaches its commit point when **all its operations** that access the database have been **executed successfully** and **the effect** of all the transaction operations on the database has been **recorded in the log**.

- After **the commit point**, the transaction **is said to be committed**, and **its effect** is assumed to be **permanently recorded** in the database.

امتي ال transaction يوصل لل commit point؟

بيقولك ان ال *transaction* بيوصل لل *commit point* لما كل العمليات الخاصه بال *transaction* تتنفذ بنجاح و ان التأثير بتاع كل العمليات ع الداتا بيز يتحفظ ف ال *system log*
وكمان بعد م ال *transaction* يكون وصل لل *commit point* بيتقال عليه انه *committed* و ان تأثيره او التغييرات الى حصلت ف الداتا بيز نتيجه العمليات الى عملها اتحفظت بشكل دائم ف الداتا بيز

ACID properties:

Atomicity: A transaction is an **atomic unit of processing**; it is either performed in its entirety or not performed at all.

Consistency preservation: A **correct execution of the transaction** must take the database **from one consistent state** to **another**.

Isolation: A transaction **should not make its updates visible to other transactions** until it is committed; this property when enforced strictly, **solves the temporary update problem**, and makes **cascading rollbacks of transactions unnecessary**

Durability (permanency): Once **a transaction changes the database** and **the changes are committed**, these changes must never be lost because of **subsequent failure**.

Transaction schedule or history: It is **the order of execution of operations** from the various transactions, when transactions are executing concurrently in an **interleaved** fashion.

- It is an **ordering of the operations of the transactions** subject to the constraint that, for each transaction Ti that participates in S, the operations of T1 in S must appear
in the same order in which they occur in T1.

ايه هو ال *Transaction schedule*
اول تعريف عندك بيقولك انه هو عمليه ترتيب العمليات م ال *transactions* المختلفه حسب تنفيذها و ده لما ال *transactions* تكون بتنفذ عملياتها بطريقه ال *interleaved* الى اتكلمنا عليها قبل كده .
تانى تعريف بيقولك انها ترتيب العمليات بتاعت ال *transactions* حسب ال *constraint* او القواعد بتاعت كل *transaction* ... بمعنى ان كل *transaction Ti* بيشترك ف ال *schedule S* العمليات بتاعته لازم تكون مكتوبه بنفس الترتيب الى بيحصل ف ال *transaction*

SHEDULES CLASSIFIED ON RECOVERABILITY:

- Recoverable schedule: In it **no** transaction needs to **be rolled back**.
- A schedule S is recoverable if **no transaction** T in S **commits** until all transactions T' that have written an item that T reads have **committed**.
- Cascadeless schedule: In it **every transaction reads only** the items that are written by committed transactions.
- Schedules requiring cascaded rollback: A schedule in which **uncommitted transactions** that read an item from a failed transaction **must be rolled back**.

- Strict schedule: A schedule in which **a transaction can neither read nor write an item X** until the last transaction that wrote X has committed.

- Serial schedule: A schedule S is serial if, for every transaction T participating in the schedule, **all the operations of T** are executed **consecutively** in the schedule.

- Serializable schedule: A schedule S is serializable if it is **equivalent** to some **serial schedule** of the same n transactions.

Result equivalent: Two schedules are called result equivalent **if they produce the same final state** of the database.

Conflict equivalent: Two schedules are said to be conflict equivalent **if the order of any two conflicting operations is the same in both schedules**.

Conflict serializable: A schedule S is said to be conflict serializable **if it is conflict equivalent to some serial schedule** S'.

• Being **serializable** means that the schedule is a correct schedule which means that:
It will leave the database in **a consistent state**.
Interleaving is appropriate and will result in a state as if the transactions were **serially executed**, and it will achieve efficiency due to **concurrent execution**.

• Serializability is hard to check due to:
- Interleaving of operations **occurs in an operating system** through some scheduler.
- It is difficult to determine beforehand **how the operations** in a schedule **will be interleaved**.

• It's **not possible** to determine when a schedule begins and when it ends.

Conflict occurs if two operations satisfy with:

- They belong to **different** transactions.
- They access the **same** item X.
- At least one of the operation is **write_item(X)**.
- EX: • r2(X) and w1(X): conflict
  - r1(X) and r2(X): Not conflict
  - w2(X) and w1(Y): Not conflict
  - r1(X) and w1(X): Not conflict

View equivalence schedule: A **less restrictive definition** of equivalence of schedules.

View serializability schedule:

- Definition of serializability **based on view equivalence**.
- A schedule is view serializability **if it is view equivalent to a serial schedule**.

Two schedules are said to be **equivalent** if the following three conditions hold:

- **The same set of transactions participates in S and S'**, and **S and S' include the same operations of those transactions**.
  For any operation $R_i(X)$ of T_i in S, if the value of X read by the operation has been written by an operation $W_j(X)$ of T_j (or if it is the original value of X before the schedule started), the same condition must hold for the value of X read by operation Ri (X) of Ti is **S'**
-

• If the operation $W_k(Y)$ of T_k is the last operation to write Y in S, then $W_k(Y)$ of T_k must also be the last operation **to write item Y in S'**.

بيقولك امتى يبقى *Two schedules* متكافئين؟
اول حاجه لما يبقى نفس ال *Transactions* موجوده ف الاتنين يعنى لو عندى *Transaction* بيعمل مثلا عمليه جمع لمتغير معين ف ال S يبقى لازم يبقى ف *Transaction* بيعمل عمليه جمع ف ال S' .. و بيقولك كمان ع انه مش يبقى فيهم نفس ال Transaction و خلاص لا زى م قولتلك قبلها لو واحد بيعمل جمع يبقى التانى يعمل جمع بردو.

طيب ركز معايا ف تانى حاجه لو عندى مثلا Transaction ف ال S بيعمل Read للمتغير X بيقولك ان لو قيمه المتغير X الى اتعملها Read ف اول Transaction اتعملها Write ف Transaction تانى ف ال S بردو .. ف كل الى حصل ده هيطبق زى م هو كده ع قيمه المتغير X الى اتعملها Read ف ال Transaction الى ف ال S'

تالت حاجه بيقولك لو عندنا Transaction بيعمل Write للمتغير Y و عمليه ال Write دى هى اخر عمليه ف ال S يبقى لازم بردو تبقى هى اخر عمليه ف ال '.

The premise behind view equivalence:

- As long as **each read operation** of a transaction reads the result **of the same write operation** in both schedules, the write operations of each transaction must produce the same results.
- **"The view":** the read operations are said to see the same view in both schedules.

Relationship between view and conflict equivalence:

• **The two are same under constrained write assumption** which assumes that if T writes X, it is constrained by the value of X it read.

• **Conflict** serializability **is stricter than view** serializability. **With unconstrained write** (**or blind write**), **a schedule that is view serializable is not necessarily conflict serializable**.

• Any **conflict** serializable schedule **is also view** serializable, but not vice versa**.**

Consider the following schedule of three transactions:

• T_1: $R_1(X)$, $W_1(X)$; T_2: $W_2(X)$; T_3: $W_3(X)$;
• Sa: $R_1(X)$; $W_2(X)$; $W_1(X)$; $W_3(X)$; C_1; C_2; C_3;

- Since T_2 and T_3 do not read the value of X, then **the** operations $W_2(X)$ and $W_3(X)$ are blind writes.
- **Sa** is **view serializable**, since it is view equivalent to the serial schedule T_1, T_2, T_3.
- However, **Sa is not conflict serializable** since it is not conflict equivalent to any serial schedule.

- **Under special semantic constraints**, schedules that are otherwise **not conflict serializable** may work **correctly**.
- **Using commutative operations of addition and subtraction** (which can be done in any order) certain **non-serializable** transactions may work **correctly**.