TRIGGER

🔁 什么时候用 `RETURN NEW` 或 `RETURN NULL`？

触发器类型	推荐返回值	含义/说明
`BEFORE INSERT/UPDATE`	`RETURN NEW` 或 `RETURN NULL`	可以修改或取消即将执行的插入/更新操作。`NULL` 表示取消操作。
`AFTER INSERT/UPDATE/DELETE`	`RETURN NEW`（或 `RETURN OLD`）	必须返回一个记录，但返回值不会被使用，只是语法要求。

1. We can write a trigger to update the enrolment number for a unit of study offering *when a student is added* (in Transcript), as shown below:

```
CREATE OR REPLACE FUNCTION updateEnrol() RETURNS trigger AS $$
BEGIN
	UPDATE UoSOffering U
	SET enrollment = enrollment+1
	WHERE U.uosCOde = NEW.uosCode AND semester = NEW.semester
		AND year = NEW.year;
	RETURN NEW;
END; $$ LANGUAGE plpgsql;

CREATE TRIGGER UpdateEnrol AFTER INSERT ON Transcript
FOR EACH ROW EXECUTE PROCEDURE updateEnrol();
```

你在判断一个unit的max enrollment 状发生了

```
create or replace function maxcountsb() returns boolean as $maxc$
begin
if ((SELECT COUNT(sailors.sid) FROM Sailors)
        + (SELECT COUNT(boats.bid) FROM boats) < 9)
    then return true;
    else
    return false;
    end if;
end;
$maxc$ language plpgsql;
```

5. 对于所有形式为 $X \xrightarrow{\rightarrow} Y$ 的多值依赖，必须满足以下 至少一个 条件：
   a. trivial MVD：
      - $X \xrightarrow{\rightarrow} Y$ 是trivial MVD，即满足以下任一条件：
        - $Y \subseteq X$，即 Y 是 X 的子集，或
        - $X \cup Y = R$，即 X 和 Y 的并集包含关系 R 的所有属性。
   b. **超键条件：**
      - X 是关系 R 的超键

employee_name	project_id	personal_phone_number
Bob	P1	047012345
Bob	P3	046098765
Bob	P1	046098765
Bob	P3	047012345
Lily	P1	045067543
Fiona	P7	043085432

Is this relation in 4NF?

No: There is at least one non-trivial multivalued dependency

存在多值依赖：$employee\_name \twoheadrightarrow Project\_id$，

而Y不是X的子集，并且X UNION Y不是R

同时$employee_name$ is not a superkey.

MVD 和 4NF 不同

Based on the data in the relation, is this a valid MVD?

UoS ↠ Tutor

Note that according to the values in the relation, the relationship between the UoS and Tutor is ***independent*** from the relationship between UoS and Textbook. *This means that the above MVD is valid.* This also implies that the Tutor of a UoS is selected independently by the school.

Assume a new Tutor, Lijun C is added for the UoS COMP9120. What must happen to maintain this independence (i.e., MVD)?

- Add one row for each different textbook with Tutor *Lijun C* of that UoS.

UoS	Textbook	Tutor
COMP9120	Silberschatz	Ying Z
COMP9120	Widom	Ying Z
COMP9120	Silberschatz	Mohammad P
COMP9120	Widom	Mohammad P
COMP9120	Silberschatz	Alan F
COMP9120	Widom	Alan F
COMP5110	Silberschatz	Ying Z
COMP5110	Silberschatz	Mohammad P
COMP9120	**Widom**	**Lijun C**
COMP9120	**Silberschatz**	**Lijun C**

ONLY IN LHS -> PART OF ALL KEYS

1. 满足atomic — From 1FN
2. No partial Dependencies

   A partial dependency is a non-trivial functional dependency $X \to Y$ in relation R, where X is a strict (proper) subset of some key for R, and Y is not part of any key.

   也就是说，不能只有主键中的一部分functional determine另外一个非主键element
3. A relation schema R is in **Third Normal Form (3NF)** if for every functional dependency $X \to Y \in F^+$, at least one of the following holds:
   a. $X \to Y$ is a **trivial** functional dependency: $Y \subseteq X$
   b. X is a **superkey** for R
   c. $Y \subseteq$ some candidate key of R

   简而言之就是："every non-key attribute is not transitively functionally dependent on a key"；**每个非key attribute不能间接依赖于key；这里是允许**函数依赖的"箭头"不从超键出发（因为c）
4. 对于所有non-trivial $X \to Y$，X 是 R 的一个**superkey**，即所有函数依赖的"箭头"必须从超键出发

b) Is the relation in 3NF?

gate->airline <u>does NOT meet</u> the 3NF restriction that either the LHS is a superkey or at least for the RHS to be part of a key. (hence cannot be in 3NF). In addition，还要检查 2NF 和 1NF, contact->name 不满足 2NF(Partial dependenct)，因为 contact 是 candidate key 中的一个；下面是老师给的例子

c) Explain whether it is a lossless-join decomposition to decompose the relation into the following:

R1(destination, departs, gate)

R2(contact, departs, pickup)

R3(gate, airline)

R4(contact, name)

这里因为 R3 和 R4 是直接和本来的 FD 相关，因此可以从 R 中删除 airline, name 这 2 个 attributes，变成 R7(dest, dep, gate, contact, pickup)；这里之所以不删除 gate 和 contact 是因为 R3 和 R7 的 intersection 为 gate 才能保证 gate 是 R3 的 superkey，对于 R4 同理。

现在来看 R7 (dest, dep, gate, contact, pickup)，他如果分解成 R1 和 R2，则它们的 intersection 是(departs)，根据现有的 FDs，我们足以推断出 departs 的 closure 不能包含所有在 R1 或 R2 里面的 attributes，因此不是 lossless 的。

Atomicity — log

- A real-world transaction is expected to happen or not happen at all
- 通过 ROLLBACK 达成

Isolation — Concurrently, interleaved

- Transactions can run concurrently; meaning their operations can be interleaved.

Incorrect Summary

T2 reads X after N is subtracted and reads Y *before* N is added: an **incorrect summary** is obtained.

Lost update

1. **事务 T1 读取某个数据项（如 `balance = 100`）。**
2. **事务 T2 也读取相同的数据项（`balance = 100`）。**
3. **T1 修改数据（如 `balance = balance + 50` → `150`），但尚未提交。**
4. **T2 也修改数据（如 `balance = balance - 20` → `80`），并先提交。**
5. **T1 提交后，`balance` 被覆盖为 `150`，而 T2 的修改（`-20`）丢失了。**

最终结果应该是 `130`（`100 + 50 - 20`），但由于并发问题，实际结果是 `150`（T2 的更新被 T1 覆盖）。

Read Uncommitted — Dirty Read

- 允许事务读取其他尚未提交事务的数据，也叫做 Dirty Read。
- PostgreSQL 不支持此隔离级别，直接转换为 Read Committed。

Read Committed — Solve **Temporary update**

- 每次查询只能读取已经提交的数据。no dirty reads.
- 防止脏读，但在同一个事务中两次读取同一行数据，可能得到不同的结果。
- **是 PostgreSQL 的默认隔离级别。**

Repeatable Read — Solve **incorrect summary problems**

- 只能"看到"在它开始之前已经提交的事务所做的更改。
- transaction 看不到：
  - 其他事务还没有提交的数据 — 避免 Read Uncommitted
  - 以及它执行期间，其他事务新提交的更改 — 避免 Read Committed

2PL

- 只能先加锁(growing)，再释放锁(Shrinking)
- 在**没有死锁**和**没有事务失败**的前提下，2PL 能够确保调度的执行是**等价于某种串行执行顺序**的，即满足可串行性要求
- Basic
  - 可以在 commit 前释放锁
- Strict
  - 只能在 commit 后释放锁

Serializable

1. 事务的可见性（MVCC）

- 事务T2开始时，会拿到一个**快照时间点**（snapshot），只看到那个时间点之前提交的所有数据版本。
- 事务T1如果在T2开始后提交了更新，**T2默认看不到T1的更新**，除非T2重启或使用特定隔离级别。

2. 那为什么串行化隔离还能避免丢失更新？

- 当T2尝试**修改**数据（写操作）时，数据库会做**序列化冲突检测**，发现T2的快照版本过旧（和T1提交的版本冲突）。
- 这时数据库会拒绝T2的提交，抛出`serialization_failure`错误。
- 事务T2必须回滚并重新执行，这样它才能看到最新版本的x，重新基于最新版本更新。

Blocking factor:

- b = block size / record size
- Record spanning: happens when b<1. Means "record size is bigger than the block size"; It is not good design when this happened

Seek time: the arm assembly is moved in or out to position a read/write head on a desired track; Go to the right track

Rotational delay: The platters spin; wait for platters spin to the right place

Transfer time: Reading/writing the data

Key approaches for physical disk-based transfer

Block transfer

- 减少 transfer time

Cylinder-based

- Principle: store relation data on the same cylinder:
- 减少 seek time

Multiple disks — RAID

- 减少 transfer 和 seek time

Disk scheduling — Elevator approach

- 减少 seek time

Prefetching/double buffering

- 减少 seek delay, Rotary delay

Total size of required pages: $105{,}264 * 4 * 1024 = 431{,}161{,}34$ bytes

Actual table size: $2{,}000{,}000 * 200 = 400{,}000{,}000$ bytes

Overhead $= \frac{(431{,}161{,}344 - 400{,}000{,}000)}{400{,}000{,}000} = 7.79\%$

Therefore, the **total number** of **index pages** is
390 + 2 + 1 = **393 index pages**

393/140,351 = **0.2% increase**

1b) Calculate the time taken to perform a table scan (i.e., linear scan) through the table Rel1.
Hint: How many pages are needed to make up the space occupied by the table? How many pages will be read from disk during a table scan?

To do a table scan we must fetch each of the 625 pages of the table; measured in seconds, this takes 625*150 = 93750 ms = 93.75 s or approximately 1.5 minutes.

Scan 的是 page，不是 record

Binary search on sorted files:

- Height $\log 140{,}351$ + additional pages containing retrieved records

Nested loop join — $O(b_R + |R| * b_S)$

Block-nested loop join — $O(b_R + b_R * b_S)$

assume that the buffer can only hold two pages for the two relations

Indexed-nested loop join — $O(b_R + (|R| * c))$

Must satisfy

- Join is an equi-join or natural join, and
  - CROSS JOIN 不满足
- an index is available on the inner table's join attribute

$\lceil log_{(B-1)}(N/B) \rceil + 1$ ← : $\lceil N / B \rceil$; CEIL(N/(B-1))

Materialization

Def

- Output of one operator written to disk and the next operator will read it from the disk.
- starting at the lowest-level.
- Produces sorted output.

Advantage:

Can always apply materialized evaluation

Disadvantages:

Costs can be quite high

Pipelining

Def

- Output of one operator is directly input to next operator
- evaluate several operations **concurrently**
- unsorted output

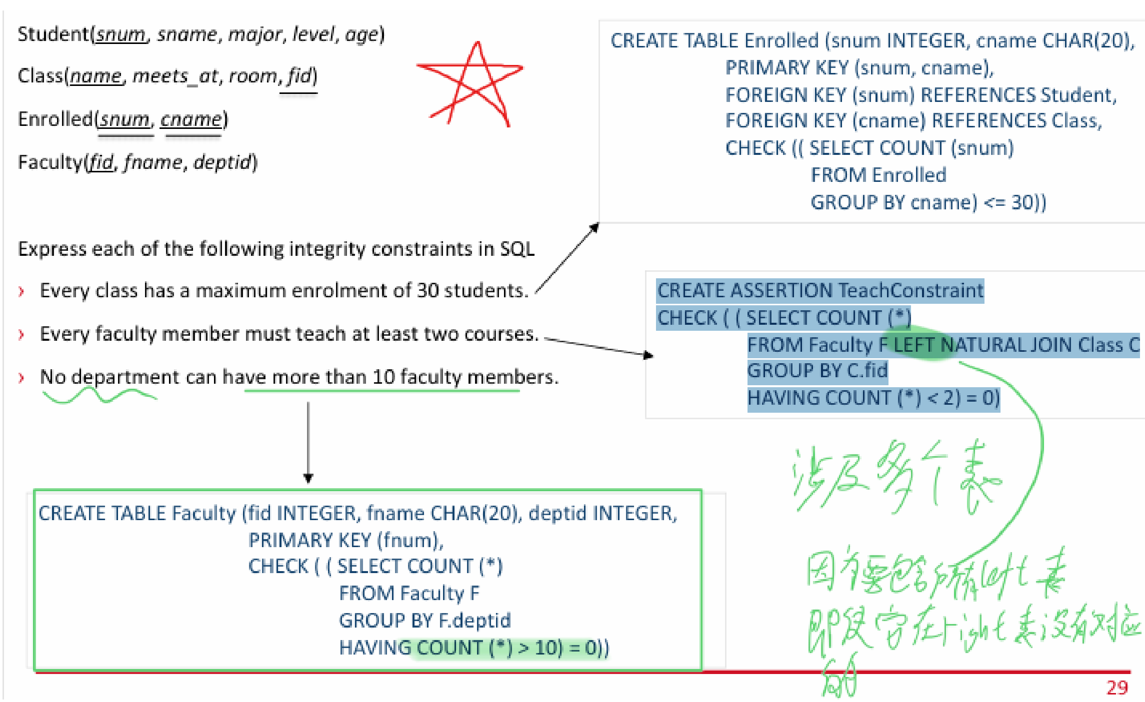
Advantage

Much cheaper than materialization:

- no need to store a temporary table

Issues:

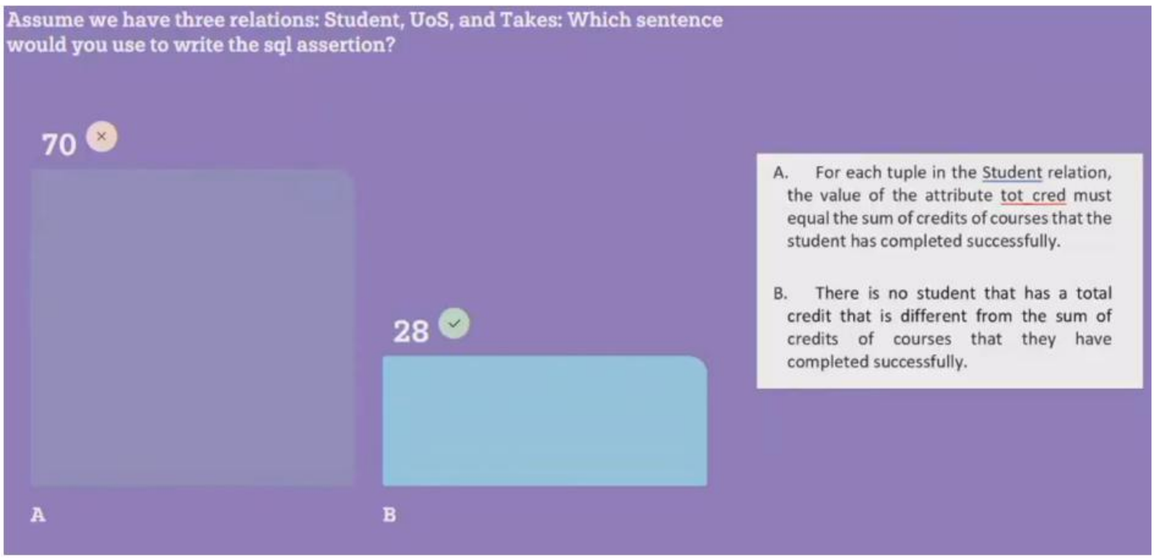
If algorithm requires sorted output, pipelining may not work well if data is not already sorted.



Insert anomaly: We cannot insert a new ProvNo to describe a doctor's speciality unless we also include a VisitID (i.e., we can't record details about a doctor unless they've made a visit).

Delete anomaly: If we delete the only record of a visit made by a doctor, we lose the information about their speciality.

Update anomaly: If a patient's address changes, we must update this address for every row in which that patient appears.

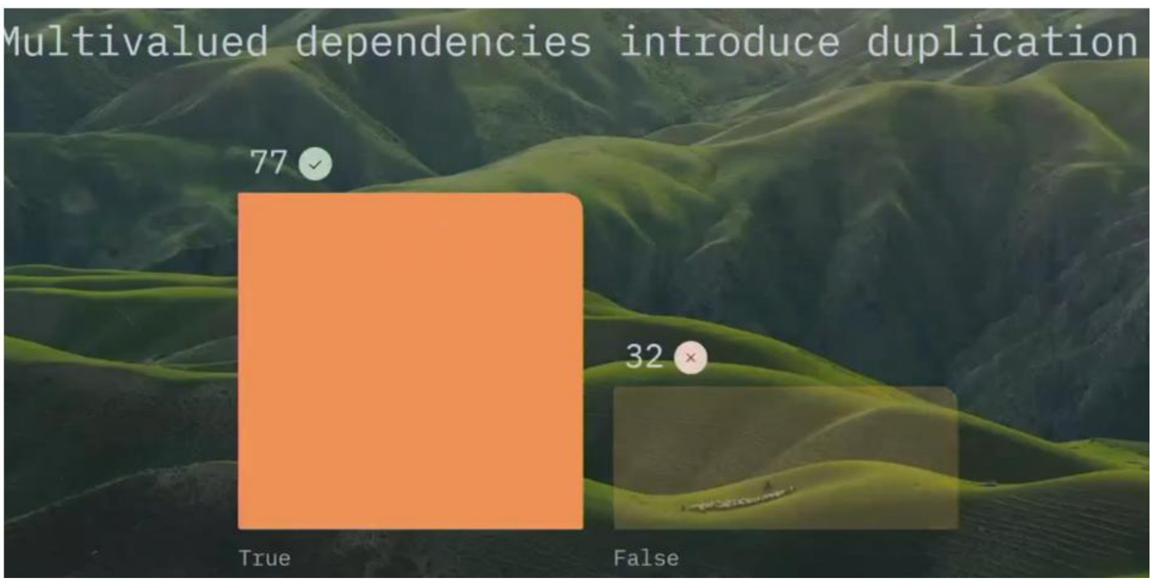


A. 针对每个学生记录，要求 `tot_cred` = 该学生完成课程学分的和

- 说的是**单个学生元组的属性值必须满足条件**。
- 就是**“对每条数据”**有条件。
- 换句话说，检查每一条学生数据是否满足规则。

B. 不存在学生的 `tot_cred` 和其完成课程学分之和不相等

- 说的是**全数据库范围内，没有一条记录违反条件**。
- 这是**一个整体性、全局性的声明**。
- 也就是说，在所有学生里，找不到不符合规则的。

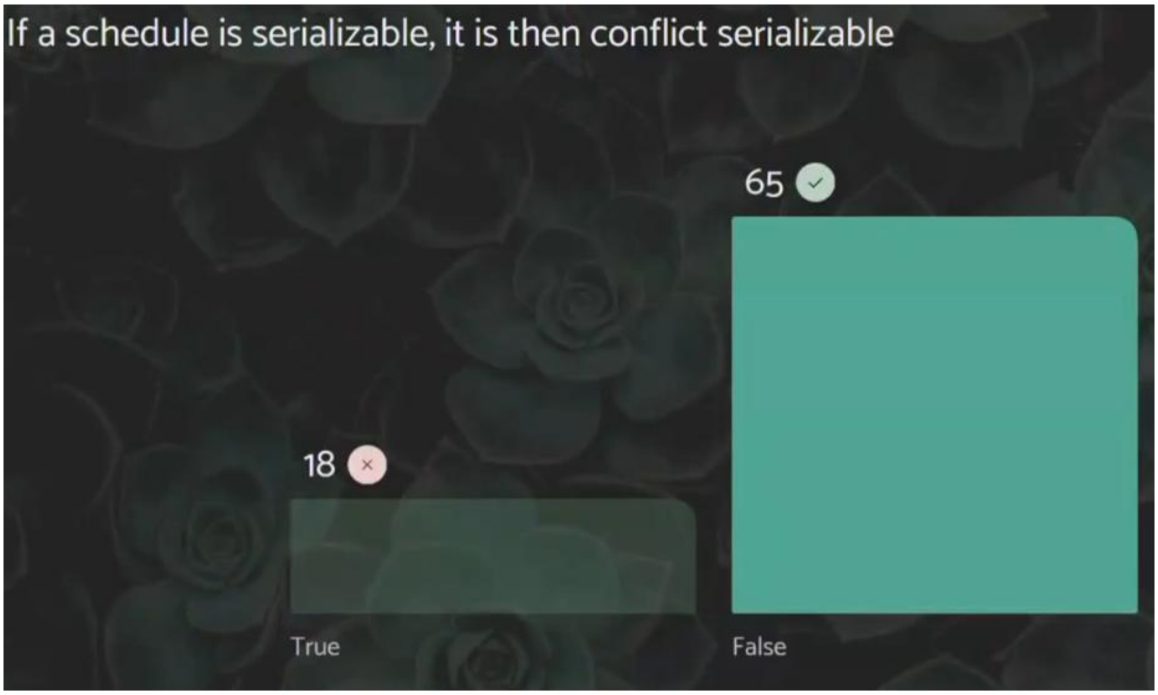


在没有 FD 的情况下判断 MVD

在一个关系中，**当某个属性 A 决定了一组值 B，而这组值与另一个属性 C 无关时**，我们说存在 **A →→ B 的多值依赖**。


Transaction logs are used for
94
14
auditing purposes
ensuring durability


Multivalued dependencies are needed to
67
41
to represent a relationship between any two set-valued attributes
establish independence between two relationships



Serializable 只看“最终结果”，即 interleaved 和 serial 的结果一样；
Conflict Serializable 既看“最终结果”，也看“操作冲突的顺序结构”。

Consider the following **interleaved** execution

```
T1: A=A-100,                                B=B+100
T2:              A=1.05*A, B=1.05*B
```

$A_F = (A_i\text{-}100)*1.05$, $B_F = B_i*1.05+100$

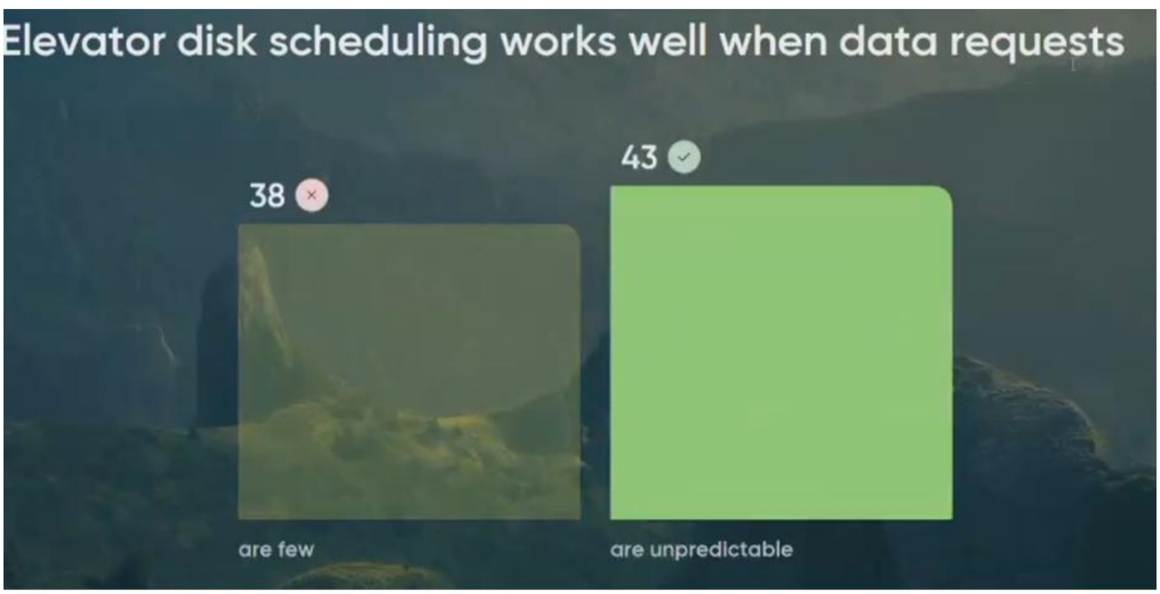
It is <u>not serializable</u>: the result of the above interleaved execution is *not the same* to *either* of the following *two serial* executions.

```
T1: A=A-100, B=B+100
T2:                           A=1.05*A, B=1.05*B
```

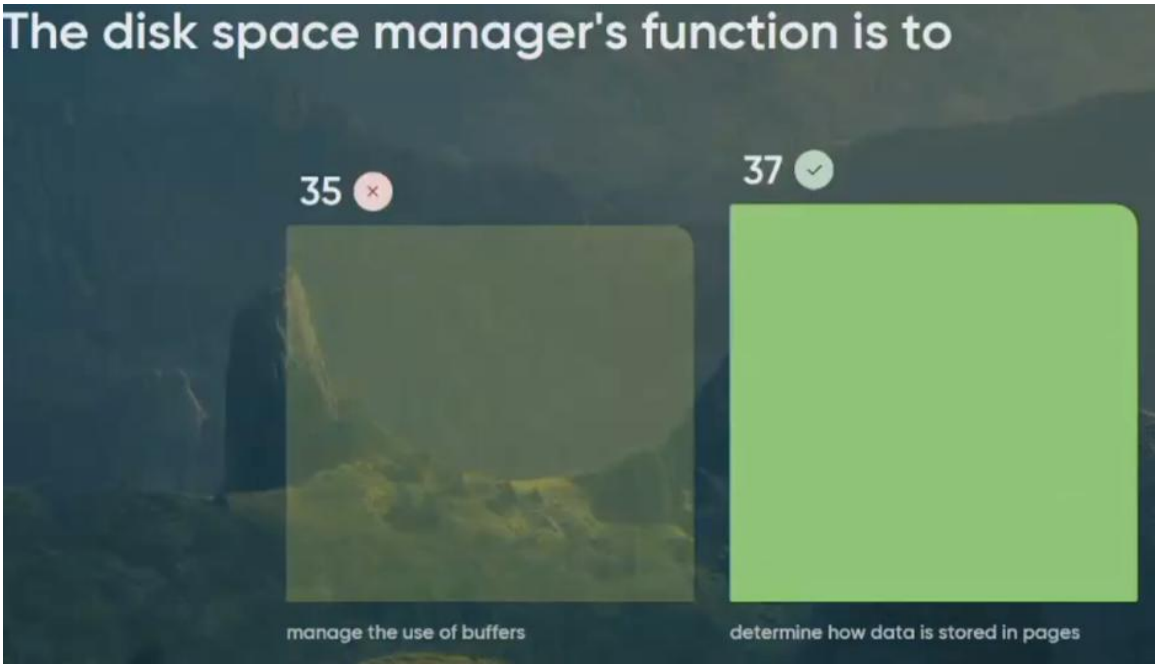
$A_F = 1.05*(A_i\text{-}100)$, $B_F = 1.05*(B_i+100)$!

```
T1:                           A=A-100, B=B+100
T2: A=1.05*A, B=1.05*B
```

$A_F =$ $(1.05*A_i)$ -100!, $B_F = (1.05*B_i)+100$



指的是磁盘请求的到达顺序、时间和位置都是**不可预知、随机变化的**情况。



Unary 指 selction，projection