

Exercise Sheet: Data Storage

Data, Data Storage, Data Collection

Exercise 1 – Hashing a New Fleet

The operator keeps the state of each bike, $\text{bike_id} \rightarrow \{\text{station}, \text{battery}, \text{status}\}$, in a hash table of m slots, with chaining: each slot holds the list of the entries that land there. The key of a bike is its number (bike B12 has key $k = 12$). As in Python, the hash of an integer is the integer itself, $\text{hash}(k) = k$, so the slot of a key k is

$$\text{hash}(k) \bmod m = k \bmod m.$$

A lookup computes the slot of the key, then walks the list of that slot and compares the key of each entry until it finds the bike.

- (Q1) A new fleet arrives, numbered by tens: B10, B20, ..., B100. Place the ten keys in a table with $m = 10$, then in a table with $m = 7$. In each table, how many entries may a lookup of B50 have to compare?
- (Q2) The keys of the new fleet are $k_j = 10j$, for $j = 1, 2, \dots$. Let d be the greatest common divisor of 10 and m . Show that $k_j \bmod m$ is always a multiple of d , and deduce the largest number of slots that the keys can reach.
- (Q3) How many slots can the keys reach for $m = 1, 7, 8, 10, 11$ and 12? Which m would you choose?
- (Q4) Real keys often follow a step: numbers by tens, even identifiers, sizes in powers of two. Why are table sizes often chosen prime?

From now on, we assume *uniform hashing*: each key lands in each of the m slots with probability $1/m$, independently of the other keys.

- (Q5) Index the keys $1, \dots, n$. For each pair $i < j$, let $X_{ij} = 1$ if keys i and j land in the same slot and $X_{ij} = 0$ otherwise, and let $N = \sum_{i < j} X_{ij}$ be the number of colliding pairs. Show that

$$\mathbb{E}[N] = \binom{n}{2} \frac{1}{m},$$

and compute it for $n = 20$ and $m = 1000$. Are the X_{ij} independent? Does it matter?

- (Q6) Insert the keys one by one. Show that the probability that no two keys collide is

$$\prod_{i=0}^{n-1} \left(1 - \frac{i}{m}\right),$$

and compute the probability of at least one collision for $n = 3$ and $m = 10$.

We consider that a table keeps its load factor $\alpha = n/m$ bounded by moving to a larger array when α gets too high.

- (Q7) Write $\mathbb{E}[N]$ in terms of n and α . How many colliding pairs do you expect for $n = 1\,000\,000$ keys and $\alpha = 1/2$?
- (Q8) Show that the expected number of other keys in the slot of a given key is $(n-1)/m$, which is less than α .
- (Q9) Do your findings contradict a lookup in $\mathcal{O}(1)$ on average? What should trigger a resize: the number of collisions or the load factor?

Exercise 2 – Two Layouts for the Trips

The operator stores its trips in a table of 1 000 trips, T1 to T1000, with 10 columns:

```
trip_id, rider_id, bike_id, start_station, end_station, start_time, end_time,
start_hour, minutes, price.
```

We assume that every value takes the same space, and that the disk reads and writes *blocks* of 100 values. Row-wise, a block holds 10 complete trips; column-wise, a block holds 100 values of a single column. The trips are stored in the order of `trip_id`, so the position of a trip is known without searching.

- (Q1) For each layout, how many values must be read to answer the query “average `minutes` per `start_hour`”? And the query “show trip T512”? Which of these values are contiguous?
- (Q2) Count the blocks read by each query in each layout. More generally, a query reads c of the C columns of all n records, in blocks of B values: how many blocks does it read in each layout? What decides the ratio?

Suppose we change the workload and the unlock app inserts a new trip at every unlock, and at every return updates `end_station`, `end_time`, `minutes` and `price` of that trip.

- (Q3) How many blocks does one insert write in each layout? One return?
- (Q4) The app makes thousands of unlocks and returns per hour; the city analysts compute the average duration per start hour once a day, over the whole history. Which layout suits each of them? How can the operator serve both?

Exercise 3 – A Night of Rebalancing

At night, a truck moves 5 bikes from station 007 (20 bikes) to station 012 (10 bikes). The database records the move as one transaction T_1 , while the city dashboard runs a transaction T_2 that displays the total number of bikes docked at the two stations. We write $R_1(007)$ when T_1 reads the record of station 007, $W_1(007)$ when it writes it, and C_1 for its COMMIT:

$$T_1 : \quad R_1(007) \quad W_1(007) \leftarrow 15 \quad R_1(012) \quad W_1(012) \leftarrow 15 \quad C_1$$

$$T_2 : \quad R_2(007) \quad R_2(012) \quad C_2 \quad (\text{displays the sum of the two values read})$$

The DBMS may interleave the operations of the two transactions. Here are three interleavings, called *schedules*, with time running downwards:

S1			S2			S3		
	T_1	T_2		T_1	T_2		T_1	T_2
1	$R_1(007)$		1	$R_1(007)$		1		$R_2(007)$
2	$W_1(007)$		2	$W_1(007)$		2	$R_1(007)$	
3		$R_2(007)$	3		$R_2(007)$	3	$W_1(007)$	
4		$R_2(012)$	4	$R_1(012)$		4	$R_1(012)$	
5		C_2	5	$W_1(012)$		5	$W_1(012)$	
6	$R_1(012)$		6	C_1		6	C_1	
7	$W_1(012)$		7		$R_2(012)$	7		$R_2(012)$
8	C_1		8		C_2	8		C_2

- (Q1) Run the two *serial* schedules, T_1 then T_2 , and T_2 then T_1 . What does the dashboard display? Recall that isolation means that concurrent transactions run as if one after the other.

- (Q2) What does the dashboard display under S1, S2 and S3? Which schedules respect isolation? In which schedules does T_2 read only committed values? Is that enough to ensure isolation?

Two operations *conflict* if they belong to different transactions, access the same record, and at least one of them is a write. The *precedence graph* of a schedule has one node per transaction, and an edge $T_i \rightarrow T_j$ whenever an operation of T_i conflicts with a later operation of T_j . One can show that a schedule can be turned into a serial one, by swapping neighbouring operations that do not conflict, exactly when its precedence graph has no cycle; the serial order is then given by a topological order of the graph.

- (Q3) Draw the precedence graphs of S1, S2 and S3, and check them against your answer to (Q2). For each schedule whose graph has no cycle, give an equivalent serial order.
- (Q4) Explain why swapping two conflicting operations may change the result, why two reads never conflict, and why a cycle in the precedence graph rules out every serial order.

A classic way for a DBMS to enforce isolation is *strict two-phase locking*, which extends the rule of the lecture to reads. A transaction locks a record before accessing it:

- a *shared* lock, to read the record: several transactions may hold one on the same record at once, since two reads do not conflict;
- an *exclusive* lock, to write the record: a transaction gets it only if no other transaction holds a lock of either kind on the record. A transaction that holds a shared lock may turn it into an exclusive one under the same condition.

A transaction that cannot get a lock waits until it is released. Locks are released only at COMMIT, all at once. The name says that a transaction first only acquires locks, then only releases them (two phases), and that the release happens at COMMIT (strict).

- (Q5) Replay the schedules that do not respect isolation with these locks: who waits, and when? What does the dashboard display, and which serial order do you get?