Databases Internals

23-09-07

Org stuff

Only labs matter for the final score. There will be quizzes.

When you work on a task `k` in project `n`, you need to:

- Create a branch named `task<k>-<mnemonic-suffix>-<last-name>`, e.g. `task0-warmup-ivanov`
- Write code + `git commit` + `git push`
- Create a PR in project `n` repo and assign a teammate as a reviewer

Recap

A **relational database** organizes data into structured tables (two-dimensional structures) with rows and columns, allowing for efficient storage, retrieval, and manipulation of data.

Indexes are data structures used in tables, improving the speed of data retrieval operations. They allow to add, remove, sort or get the rows without scanning the whole table. Common types used for implementing indexes are B-tree (balanced tree) and hash table.

Basic concepts of transactions:

- *Atomicity*. Transaction is treated as a single, indivisible unit of work If a transaction fails at any point, it is rolled back to its previous state.
- *Consistency*. This means that the integrity constraints of the database (e.g., primary keys, foreign keys) should not be violated during a transaction.
- *Isolation*. Changes made by one transaction should not be visible to other transactions until the first transaction is committed. This ensures that concurrent transactions do not interfere with each other.

- *Durability*. Once a transaction is committed, its changes should be permanent and survive system failures. The database should be able to recover to a consistent state after a crash.

SQL commands

SQL is a domain-specific language used to interact with relational databases.

Example of creating tables:

```sql
CREATE TABLE customers (
    customer_id INT PRIMARY KEY,
    customer_name VARCHAR(50)
);

CREATE TABLE orders (
    order_id INT PRIMARY KEY,
    customer_id INT,
    order_date DATE,
    total_amount DECIMAL(10, 2)
);

-- Inserting some sample data into the "customers" table
INSERT INTO customers (customer_id, customer_name)
VALUES
    (1, 'Alice'),
    (2, 'Bob'),
    (3, 'Charlie'),
    (4, 'David');

-- Inserting some sample data into the "orders" table
INSERT INTO orders (order_id, customer_id, order_date, total_amount)
VALUES
    (101, 1, '2023-01-05', 250.00),
    (102, 2, '2023-01-10', 120.50),
    (103, 1, '2023-01-15', 300.75),
    (104, 3, '2023-01-20', 450.20);
```

Customers

customer_id	customer_name
1	Alice
2	Bob
3	Charlie
4	David

Orders

order_id	customer_id	order_date	total_amount
101	1	2023-01-05	250
102	2	2023-01-10	120.5
103	1	2023-01-15	300.75
104	3	2023-01-20	450.2

`JOIN` command is used to combine rows from two or more tables based on a related column between them.

- `INNER JOIN` returns the rows that have matching values in both tables:

```sql
SELECT orders.order_id, customers.customer_name
FROM orders
INNER JOIN customers ON orders.customer_id = customers.customer_id;
```

```
| order_id | customer_name |
|----------|---------------|
| 101      | Alice         |
| 102      | Bob           |
| 103      | Alice         |
| 104      | Charlie       |
```

- `LEFT JOIN` returns all rows from the left table and the matched rows from the right table. If there is no match in the right table, `NULL` value is used:

```sql
SELECT customers.customer_name, orders.order_id
FROM customers
LEFT JOIN orders ON customers.customer_id = orders.customer_id;
```

```
| customer_name | order_id |
|---------------|----------|
| Alice         | 101      |
| Bob           | 102      |
| Alice         | 103      |
| Charlie       | 104      |
| David         | NULL     |
```

- `RIGHT JOIN` (or `RIGHT OUTER JOIN`) returns all rows from the right table and the matched rows from the left table. If there is no match in the left table, NULL values are returned for columns from the left table:

```sql
SELECT customers.customer_name, orders.order_id
FROM customers
RIGHT JOIN orders ON customers.customer_id = orders.customer_id;
```

```
| customer_name | order_id |
|---------------|----------|
| Alice         | 101      |
| Bob           | 102      |
| Alice         | 103      |
| Charlie       | 104      |
```

- `FULL OUTER JOIN` returns all rows when there is a match in either the left or right table. If there is no match, NULL values are returned for columns from the table with no match:

```sql
SELECT customers.customer_name, orders.order_id
FROM customers
FULL OUTER JOIN orders ON customers.customer_id = orders.customer_id;
```

```
| customer_name | order_id |
|---------------|----------|
| Alice         | 101      |
| Bob           | 102      |
| Alice         | 103      |
| Charlie       | 104      |
| David         | NULL     |
```

The `WHERE` clause is used to filter rows based on specified conditions:

```sql
SELECT orders.order_id, customers.customer_name
FROM orders
INNER JOIN customers ON orders.customer_id = customers.customer_id
WHERE orders.order_date >= '2023-01-12';
```

```
| order_id | customer_name |
|----------|---------------|
| 103      | Alice         |
| 104      | Charlie       |
```

Secondary storage

Primary storage is what CPU can reach ”directly”, e.g. cache or RAM. Secondary storage is accessed with controllers and it is where all our data resides permanently.

There are several data storage characteristics:

- how much data can be stored (capacity)
- how fast a random access is (random access latency)
- how much data per second can be read or written (transfer rate)
- what happens if electricity switches off (volatility)
- how much does it cost to store 1Gb (price)

	Capacity	Latency	T/Rate	Price
L1-L3	32 kb - 96 Mb	1-10 ns	400-3000 Gb/s	$ ..50k/Gb
RAM	2 Gb-40 Tb	10-20 ns	10-20 Gb/s	$ 5/Gb
HDD	..30 Tb	6-20 ms	100-300 Mb/s	$ 0.015/Gb
SSD	..30 Tb	0.07..0.25 ms	0.5-5 Gb/s	$ 0.03-0.1/Gb
Tape	..20 Tb	:)	400 Mb/s	$ 0.005/Gb

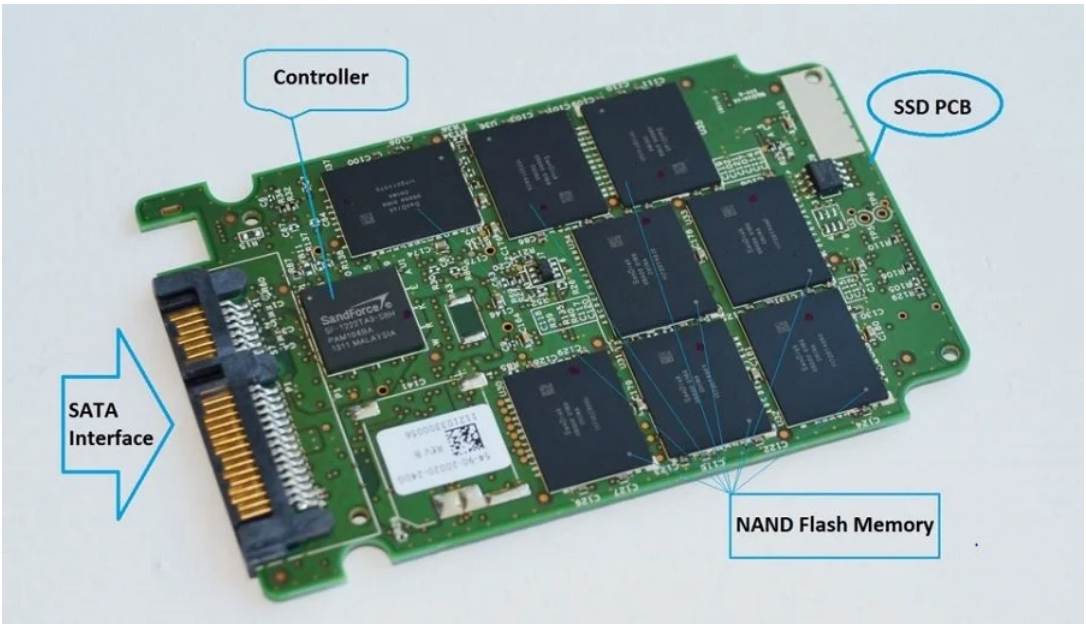
HDD

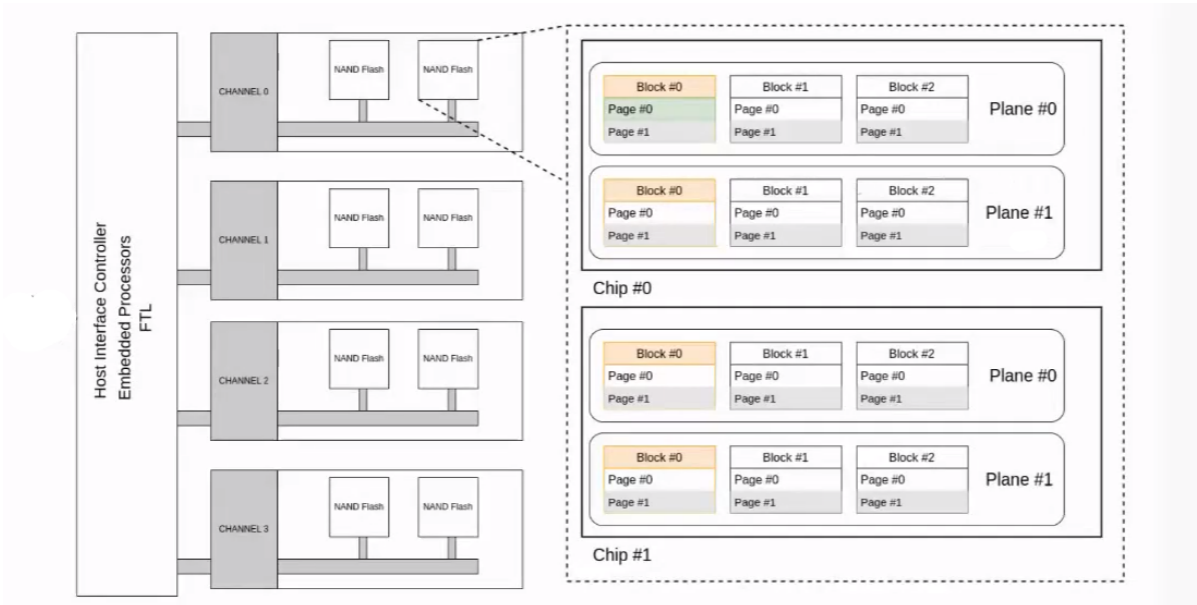
A disk consists of several *platters*. A *track* is a circle on a platter. Tracks located one above other on different platters form a *cylinder*. A *sector* is a fragment of the track.

In the HDD, data is stored on a **rotating magnetic disk** which is divided into tracks, sectors and cylinders. An electromagnet in the read/write head charges the disk's surface with either a positive or negative charge, this is how binary 1 or 0 is represented.

The **read/write head** is then capable of detecting the magnetic charges left on the disk's surface, this is how data is read. A circuit board carefully co-ordinates rotating the disk and swinging the **actuator arm** to allow the read/write head to access any location very quickly.

SSD





The **NAND flash memory** is the primary storage medium in the SSD. It's a type of non-volatile memory that stores data as electrical charges in memory cells.

NAND flash memory chips are circuits that contain NAND flash memory. The SSD contains multiple chips, often arranged in a grid-like fashion.

The NAND flash memory is organized into **pages**. The data is read from or written to these pages. Pages are grouped together into larger units called **blocks**.

Channels are pathways through which data is transferred between the SSD controller and the NAND flash memory chips. The SSD typically has multiple channels, and the data can be stripped over them. Thus, it can be read from or written to the chips in parallel.

When it comes to reading data, random access in SSD is rather fast. But if many small reads are executed, there is a high chance that all of them are stored in the same channel.

When it comes to writing, *there are no in-place modifications*. Instead, a new page is created, and the old one is marked invalid and erased. This is because NAND flash memory cells can be written to a limited number of times before they wear out. Spreading out the write and erase operations across different memory cells helps prolong the lifespan of the SSD.

When it comes to erasing pages, *a block is a minimal-erase unit*. In other words, you can't erase individual pages within a block; you must erase the entire block. Garbage collection identifies blocks that contain invalid pages. It copies the valid data to new blocks, and erases the old blocks.

As you can see, large reads/writes are better than small ones.