Cos’è un CRUD

Programmazione

Che cos’è un CRUD (Create, Read, Update, Delete)

Un CRUD è un insieme di quattro operazioni fondamentali utilizzate in qualsiasi sistema informatico che gestisce dati persistenti.

L’acronimo identifica i quattro comportamenti minimi che un’applicazione deve offrire per interagire con un archivio dati: Create (creare nuovi record), Read (leggere o recuperare dati), Update (aggiornare o modificare dati esistenti), e Delete (eliminare dati).

Queste quattro azioni costituiscono il ciclo base della manipolazione delle informazioni ed è il motivo per cui quasi ogni software gestito tramite database o API segue questo schema. In altre parole, un CRUD rappresenta la “grammatica” con cui un sistema tratta i dati.

Come si usa nel contesto IT e dei database

Nel mondo IT, un CRUD viene implementato tipicamente come insieme di endpoint API, funzioni applicative o query SQL che permettono a un'applicazione di interagire con un database.

Per esempio, in SQL, il mapping naturale del CRUD è: INSERT → Create, SELECT → Read, UPDATE → Update, DELETE → Delete.

Queste operazioni costituiscono la base di qualunque servizio che memorizza informazioni, dai sistemi gestionali ai microservizi basati su REST.

A livello architetturale, il CRUD determina la struttura dei controller, la progettazione delle interfacce utenti, e persino gli schemi di persistenza: ogni tabella o entità del dominio avrà generalmente un set di operazioni CRUD associato, che diventa la base su cui costruire logiche più complesse come validazioni, controlli di accesso o automazioni lato backend.

CREATE – Creare un nuovo dato

L’operazione Create serve ad aggiungere un nuovo record al sistema. In un vero backend potresti fare una chiamata POST verso un’API; in un database useresti INSERT.

In JavaScript, se simuli un archivio dati con un array, creare significa semplicemente aggiungere un oggetto alla lista.

L’importante è garantire che il nuovo elemento abbia un’identità unica (es. un id), dato che servirà per tutte le altre operazioni CRUD.

CREATE – Creare un nuovo dato

```
let utenti = [];

// CREATE
function createUser(nome) {
  const nuovoUtente = {
    id: Date.now(),   // simuliamo un ID unico
    nome: nome
  };

  utenti.push(nuovoUtente);
  return nuovoUtente;
}

console.log(createUser("Mirko"));
```

READ – Leggere o recuperare dati

L’operazione Read serve a ottenere dati dal sistema. In un DB è l’equivalente di una SELECT, mentre in un’applicazione web è spesso una chiamata GET.

Nel nostro caso, leggere significa semplicemente restituire l’intera lista o cercare un singolo elemento tramite una condizione (es. “trova utente con id X”).

L’operazione Read non modifica nulla: è puramente consultiva.

READ – Leggere o recuperare dati

```
// READ - ottenere tutti gli utenti
function getAllUsers() {
  return utenti;
}

// READ - trovare un utente per ID
function getUserById(id) {
  return utenti.find(u => u.id === id);
}

console.log(getAllUsers());
```

UPDATE – Aggiornare dati esistenti

L’operazione Update modifica un record già presente senza crearne uno nuovo.

In un contesto SQL è equivalente alla UPDATE, mentre in un API sarebbe una chiamata PUT o PATCH.

La logica fondamentale è:

1. **cercare il dato da modificare (es. tramite ID),**
2. **cambiare solo i campi desiderati,**
3. **salvare nuovamente l’oggetto aggiornato.**

UPDATE – Aggiornare dati esistenti

```
// UPDATE
function updateUser(id, nuovoNome) {
  const utente = utenti.find(u => u.id === id);
  if (!utente) return null;

  utente.nome = nuovoNome;
  return utente;
}

// esempio di utilizzo:
const idDaAggiornare = utenti[0].id;
console.log(updateUser(idDaAggiornare, "Mirko Updated"));

```

DELETE – Eliminare un dato

La Delete rimuove un record dal sistema.

Nel mondo SQL è una DELETE, mentre nelle API è spesso una chiamata DELETE verso un endpoint che rappresenta la risorsa.

In JavaScript eliminare un elemento da una lista significa filtrarlo oppure rimuoverlo tramite indice.

È importante restituire un valore che indichi se la cancellazione è avvenuta oppure no.

DELETE – Eliminare un dato

```
// DELETE
function deleteUser(id) {
  const lunghezzaOriginale = utenti.length;
  utenti = utenti.filter(u => u.id !== id);

  return utenti.length < lunghezzaOriginale; // true se eliminato
}

// esempio di utilizzo:
console.log(deleteUser(idDaAggiornare));

// true se l'utente è stato cancellato
```

Problemi strutturali e architetturali nei sistemi CRUD

Un CRUD tradizionale tende a essere strettamente legato alla struttura del database: ogni entità ha le sue quattro operazioni e l'applicazione spesso riflette direttamente lo schema SQL.

Questa semplicità, se da un lato facilita lo sviluppo iniziale, dall'altro diventa un limite quando la complessità cresce. Il modello CRUD, essendo stateful e basato su mutazioni dirette dei record, può creare forte accoppiamento tra livelli (UI ↔ controller ↔ database).

La mancanza di astrazione porta a logiche duplicate, difficoltà nel versioning dei dati, problemi di scalabilità nelle query e un aumento della complessità nel mantenere consistenza e integrità dei dati in sistemi distribuiti.

Inoltre, il pattern CRUD non gestisce nativamente concetti come transazioni complesse, eventi di dominio, o vincoli cross-entity, che diventano critici in architetture enterprise.

Problemi di sicurezza, concorrenza e performance

Dal punto di vista operativo, un CRUD può diventare un punto debole se non vengono applicate misure di sicurezza adeguate.

Operazioni come Update e Delete sono intrinsecamente distruttive e richiedono rigorosi controlli di permessi, validazione dell'input e prevenzione di attacchi come SQL injection o mass assignment.

Allo stesso modo, in ambienti multi-utente, la concorrenza può generare race condition: due processi che aggiornano lo stesso record potrebbero sovrascriversi a vicenda se non si usano lock, versioning o transazioni.

Anche le performance possono degradare rapidamente: operazioni di lettura senza indicizzazione, delete su tabelle molto grandi, o update frequenti possono bloccare risorse e aumentare la latenza.

Senza caching, paginazione e strategie di ottimizzazione, un CRUD può diventare un collo di bottiglia per l'intera applicazione.


Buon DAVANTE a tutti