ITIS Informatica - Quinta Superiore

---

1. METODOLOGIA PER SVOLGERE LE QUERY

1.1 Approccio Sistematico

Processo in 4 fasi:

1. **Analisi del problema**
   - Identificare le tabelle coinvolte
   - Determinare i campi necessari nel risultato
   - Individuare le condizioni di filtro
2. **Definizione delle relazioni**
   - Stabilire le chiavi primarie e esterne
   - Determinare il tipo di JOIN necessario
3. **Costruzione della query**
   - Struttura base: SELECT → FROM → WHERE → GROUP BY → HAVING → ORDER BY
   - Aggiungere funzioni e sottoquery se necessarie
4. **Verifica e ottimizzazione**
   - Controllo logico del risultato
   - Verifica della sintassi

1.2 Pattern Ricorrenti negli Esami

Dalle tracce analizzate emergono questi pattern tipici:

Elenchi semplici:

```
SELECT campo1, campo2
FROM tabella
WHERE condizione
ORDER BY campo;
```

Elenchi con JOIN:

```
SELECT t1.campo, t2.campo
FROM tabella1 t1
```

```
JOIN tabella2 t2 ON t1.fk = t2.pk
WHERE condizione;
```

Conteggi e aggregazioni:

```
SELECT campo, COUNT(*) as totale
FROM tabella
GROUP BY campo
ORDER BY totale DESC;
```

---

2. FUNZIONI SQL FONDAMENTALI

2.1 Operatore BETWEEN

Sintassi: `campo BETWEEN valore1 AND valore2`

Utilizzo tipico con date:

```
-- Esempio da traccia 2019: film visualizzati in un intervallo
SELECT titolo, COUNT(*) as visualizzazioni
FROM film f, visualizzazioni v
WHERE f.codice = v.fk_codice_film
AND v.data BETWEEN '2019-04-01' AND '2019-04-04'
GROUP BY titolo;
```

Equivalenza logica:

- `BETWEEN` è equivalente a `>= valore1 AND <= valore2`
- Include sempre i valori estremi

2.2 Funzione YEAR() e Gestione Date

Estrazione dell'anno:

```
-- Esempio: prodotti dell'anno corrente
SELECT nome_prodotto
FROM prodotti
WHERE YEAR(data_produzione) = YEAR(CURDATE());
```

Operazioni con date comuni:

```
-- Data corrente
SELECT CURDATE();
```

```sql
-- Anno specifico
WHERE YEAR(data_campo) = 2023

-- Confronto date
WHERE data_inizio > '2023-01-01'
```

2.3 Clausola HAVING

Differenza fondamentale con WHERE:

- **WHERE:** filtra righe prima dell'aggregazione
- **HAVING:** filtra gruppi dopo l'aggregazione

Esempio pratico:

```sql
-- Traccia 2023: conteggio utilizzi videogiochi
SELECT videogioco_id, COUNT(*) as utilizzi
FROM utilizzi_classe
GROUP BY videogioco_id
HAVING COUNT(*) > 10;  -- Solo giochi usati più di 10 volte
```

Pattern tipico negli esami:

```sql
-- Media con filtro su aggregazione
SELECT regione, AVG(quantita) as media_produzione
FROM produzione_miele
GROUP BY regione
HAVING AVG(quantita) > 1000;
```

---

3. SOTTOQUERY (SUBQUERY)

3.1 Tipologie Principali

A. Sottoquery scalari (restituiscono un valore):

```sql
-- Esempio: utenti sopra la media
SELECT nome, punteggio
FROM utenti
WHERE punteggio > (
    SELECT AVG(punteggio)
    FROM utenti
);
```

B. Sottoquery con IN/NOT IN:

```sql
-- Esempio da traccia: utenti che non hanno mai visualizzato film
SELECT nome, cognome
FROM utenti
WHERE nome_utente NOT IN (
    SELECT fk_nome_utente
    FROM visualizzazioni
);
```

C. Sottoquery correlate:

```sql
-- Esempio: studenti con più assenze della media della loro classe
SELECT s.nome, s.cognome
FROM studenti s
WHERE (
    SELECT COUNT(*)
    FROM assenze a
    WHERE a.studente_id = s.id
) > (
    SELECT AVG(conta_assenze)
    FROM (
        SELECT COUNT(*) as conta_assenze
        FROM assenze a2, studenti s2
        WHERE a2.studente_id = s2.id
        AND s2.classe = s.classe
        GROUP BY s2.id
    ) AS media_classe
);
```

3.2 Operatori con Sottoquery

EXISTS / NOT EXISTS:

```sql
-- Verifica esistenza correlata
SELECT p.nome
FROM progetti p
WHERE EXISTS (
    SELECT 1
    FROM team t
    WHERE t.progetto_id = p.id
    AND t.ruolo = 'capo_progetto'
);
```

ALL / ANY:

```sql
-- Maggiore di tutti i valori
WHERE valore > ALL (SELECT campo FROM tabella WHERE condizione)

-- Maggiore di almeno un valore
WHERE valore > ANY (SELECT campo FROM tabella WHERE condizione)
```

3.3 Strategia per Sottoquery Complesse

1. **Isolamento del problema interno**
   - Scrivere prima la sottoquery in modo indipendente
   - Verificarne il risultato
2. **Integrazione graduale**
   - Aggiungere la sottoquery alla query principale
   - Testare il risultato combinato
3. **Ottimizzazione**
   - Considerare alternative con JOIN quando possibile
   - Valutare l'uso di viste temporanee per query molto complesse

---

4. ESEMPI PRATICI DA TRACCE D'ESAME

Esempio 1: Query con Date e HAVING (Traccia 2023)

```sql
-- Apicoltori che producono miele DOP in una regione
SELECT a.nome, a.cognome, m.denominazione
FROM apicoltori a, apiari ap, mieli m
WHERE a.id = ap.apicoltore_id
AND ap.miele_id = m.id
AND m.tipologia = 'DOP'
AND ap.regione = 'Toscana'
AND YEAR(ap.anno_produzione) = 2023;
```

Esempio 2: Sottoquery con NOT IN (Traccia 2019)

```sql
-- Film mai visualizzati
SELECT f.titolo
FROM film f
WHERE f.codice NOT IN (
    SELECT DISTINCT v.fk_codice_film
    FROM visualizzazioni v
    WHERE v.fk_codice_film IS NOT NULL
);
```

Esempio 3: Aggregazione con HAVING (Traccia 2024)

```sql
-- Palestre con più di 50 prenotazioni
SELECT p.nome, COUNT(*) as totale_prenotazioni
FROM palestre p, prenotazioni pr
WHERE p.id = pr.palestra_id
AND YEAR(pr.data_prenotazione) = 2024
GROUP BY p.id, p.nome
HAVING COUNT(*) > 50
ORDER BY totale_prenotazioni DESC;
```

---

5. CHECKLIST PER L'ESAME

Verifica Sintattica

- [ ] Tutte le tabelle sono specificate nel FROM
- [ ] I JOIN sono corretti con le chiavi appropriate
- [ ] Le funzioni aggregate sono usate con GROUP BY quando necessario
- [ ] HAVING è usato solo dopo GROUP BY
- [ ] Le sottoquery sono chiuse correttamente con parentesi

Verifica Logica

- [ ] Il risultato risponde alla domanda posta
- [ ] I filtri WHERE sono logicamente corretti
- [ ] L'ordinamento ORDER BY è appropriato
- [ ] Le condizioni di JOIN evitano prodotti cartesiani

Ottimizzazione

- [ ] Uso di alias per migliorare la leggibilità
- [ ] Sottoquery sostituibili con JOIN più efficienti
- [ ] Eliminazione di campi non necessari nel SELECT