ORACLE SQL | MANUALE PER STUDENTI E COLLEGHI

# UNINA-SPOTIFY

## Capire ogni query

18 richieste svolte e motivate, passo passo.

Scelte corrette, alternative e insidie da riconoscere.

---

6 artisti  11 studenti  18 brani  72 ascolti

Query eseguite su Oracle Database 21c XE.

Valori, duplicati e ordinamento controllati sui dati originali.

Confronto con le soluzioni del docente e fonti Oracle.

Antonio Ferone

antonioferone.it

22 ottobre 2026 | Edizione didattica commentata

# Come usare questo manuale

Non imparare le query a memoria: impara a collegare le parole della traccia alla struttura del risultato.

Per ogni richiesta trovi due pagine: prima la query commentata, poi ragionamento, motivazioni, alternative, esempio e autoverifica. La richiesta 17 è divisa in 17a e 17b.


| Richiesta | Argomento | Pagine |
|---|---|---|
| 1 | Brani rock oltre 240 secondi | 5-6 |
| 2 | Ascolti completi di Data Science | 7-8 |
| 3 | Ascolti di aprile 2026 | 9-10 |
| 4 | Numero e durata media per genere | 11-12 |
| 5 | Gruppi ascoltati da più di cinque studenti | 13-14 |
| 6 | Percentuale media di ascolto sotto il 90% | 15-16 |
| 7 | Tutti i brani, anche senza ascolti | 17-18 |
| 8 | Studenti e ascolti per corso | 19-20 |
| 9 | Brano più lungo | 21-22 |
| 10 | Brano con più ascolti | 23-24 |
| 11 | Studenti con più ascolti | 25-26 |
| 12 | Studenti che hanno ascoltato artisti italiani | 27-28 |
| 13 | Brani sopra la media del proprio genere | 29-30 |
| 14 | Studenti che non hanno mai ascoltato Rap | 31-32 |
| 15 | Studenti che hanno ascoltato tutti i brani dei Pink Floyd | 33-34 |
| 16 | Vista di dettaglio e riepilogo Data Science | 35-36 |
| 17a | Pink Floyd ma non Queen | 37-38 |
| 17b | Sia Queen sia AC/DC | 39-40 |
| 18 | Ascolti per tutti i mesi del 2026 | 41-42 |


Uso pratico: nella connessione UNINA, apri il file SQL commentato. F5 esegue lo script; Ctrl+Invio esegue l'istruzione corrente. Per la richiesta 16 esegui la creazione della vista prima della SELECT.

Traccia: professor Moscato. Guida ragionata: Antonio Ferone.

# Prima del codice: le relazioni

Una domanda SQL può fallire anche con una sintassi valida. Il primo controllo è capire che cosa rappresenta una riga in ogni tabella.

| Tabella | Una riga rappresenta | Chiave |
|---|---|---|
| ARTISTI | Un artista o gruppo musicale | id_artista |
| STUDENTI | Una persona iscritta | id_studente |
| BRANI | Un brano del catalogo | id_brano |
| ASCOLTI | Un evento di ascolto | id_ascolto |

## I collegamenti da tenere sempre a vista

```
BRANI.artista    = ARTISTI.id_artista
ASCOLTI.brano    = BRANI.id_brano
ASCOLTI.studente = STUDENTI.id_studente
```

Un artista può avere molti brani. Un brano può ricevere molti ascolti. Uno studente può effettuare molti ascolti. Per questo un JOIN con ASCOLTI può ripetere la riga dello studente o del brano: non è automaticamente un errore.

## L'ordine logico utile per ragionare

FROM e JOIN/ON -> WHERE -> GROUP BY -> HAVING -> SELECT -> DISTINCT -> ORDER BY

È un modello logico per capire il significato, non la sequenza fisica obbligatoria del motore. L'ottimizzatore può scegliere piani diversi che conservano la stessa semantica.

Prima colleghi e filtri le righe; poi formi eventuali gruppi e filtri i gruppi; infine scegli le colonne e ordini. Se cambia la granularità - per esempio da ascolto a studente - devi renderlo esplicito con un raggruppamento o con una condizione di esistenza.

# Una cassetta degli attrezzi

Scegli l'operatore per ciò che deve significare la risposta. Una forma più breve non è necessariamente più corretta; due forme differenti possono essere equivalenti.

### Voglio tutte le righe che rispettano un filtro

WHERE su colonne delle singole righe. Le parentesi chiariscono come combinare AND e OR.

### Voglio un riepilogo per categoria

GROUP BY e funzioni aggregate. HAVING filtra il risultato dei gruppi, WHERE filtra prima le righe.

### Voglio conservare anche chi non ha eventi

LEFT JOIN partendo dall'insieme completo. Per contare gli eventi usa una chiave non nulla del lato destro.

### Voglio almeno un evento / nessun evento

EXISTS / NOT EXISTS con la correlazione alla riga esterna. La domanda è di presenza, non di conteggio.

### Voglio tutti gli elementi richiesti

Doppio NOT EXISTS: non deve esistere un elemento richiesto che manchi. Attenzione agli insiemi vuoti.

### Voglio chi è nel primo insieme ma non nel secondo

MINUS. Per chi è in entrambi, INTERSECT. L'operazione confronta righe complete compatibili.

### Voglio i massimi senza perdere i pari merito

Confronta con MAX, oppure usa consapevolmente una funzione analitica o FETCH ... WITH TIES.

> Tre contatori diversi: COUNT(*) conta righe; COUNT(colonna) conta valori non NULL; COUNT(DISTINCT colonna) conta valori distinti non NULL. Non sono intercambiabili.

Gli esempi sono ragionati per Oracle 21c XE. LIMIT non è la sintassi Oracle. SQL ammette più soluzioni corrette: qui motiviamo soprattutto significato, leggibilità e aderenza alla traccia; non promettiamo prestazioni senza un piano di esecuzione.

RICHIESTA 1 | QUERY COMMENTATA

# Brani rock oltre 240 secondi

Mostrare titolo, artista e durata dei brani di Rock progressivo oppure Hard rock che superano 240 secondi, dal più lungo.

Una riga del risultato = un brano che supera entrambi i filtri richiesti.

```
-- Scegli le tre informazioni richieste, senza aggregare.
SELECT B.titolo, AR.nome AS artista, B.durata_secondi
-- BRANI contiene titolo, genere e durata.
FROM BRANI B
-- La chiave esterna collega il brano al nome dell'artista.
JOIN ARTISTI AR ON AR.id_artista = B.artista
-- Il genere deve essere uno dei due valori ammessi.
WHERE B.genere IN ('Rock progressivo', 'Hard rock')
-- La durata deve superare 240 per entrambi i generi.
  AND B.durata_secondi > 240
-- DESC = dal più lungo al più breve.
ORDER BY B.durata_secondi DESC;
```

Risultato verificato: 4 brani: Comfortably Numb 382; Wish You Were Here 334; Thunderstruck 292; Back in Black 255.

RICHIESTA 1 | IL RAGIONAMENTO

# Perché si scrive così

Leggi il percorso come una trasformazione del problema, non come una sequenza da memorizzare.

### 1. Parti dal dato che cerchi

Titolo, genere e durata sono in BRANI. Il nome leggibile dell'artista è invece in ARTISTI: la sola chiave numerica B.artista non basta.

### 2. Collega le chiavi corrette

B.artista è una chiave esterna verso AR.id_artista. Il JOIN associa ogni brano al suo artista, senza incrociare tutti i brani con tutti gli artisti.

### 3. Applica i due filtri insieme

IN significa che il genere appartiene all'elenco dei due valori. AND richiede anche durata_secondi > 240: 240 esatti non sono sufficienti.

### 4. Ordina solo alla fine

DESC mette le durate maggiori prima delle minori. L'ordinamento non filtra e non cambia quali brani entrano nel risultato.

## Scelte e alternative

IN invece di due uguaglianze. B.genere = 'Rock progressivo' OR B.genere = 'Hard rock' è equivalente, purché le due condizioni siano racchiuse tra parentesi prima dell'AND sulla durata. IN rende questo gruppo più evidente.

JOIN esplicito invece della virgola. La soluzione del docente usa FROM BRANI B, ARTISTI AR con la relazione nel WHERE: è valida. JOIN ... ON separa visivamente le relazioni tra tabelle dai filtri richiesti dalla traccia.

Nessun GROUP BY e nessun DISTINCT. Non stiamo riassumendo più brani in una sola riga e non abbiamo duplicazioni da eliminare. Raggruppare per artista perderebbe il dettaglio dei singoli brani.

> Esempio sui dati: Comfortably Numb dura 382 secondi ed è Rock progressivo: entra. Another Brick in the Wall, Part 2 è dello stesso genere ma dura 238 secondi: resta fuori.

Errore da evitare. Senza parentesi, genere = 'Rock progressivo' OR genere = 'Hard rock' AND durata > 240 applica la durata soltanto al secondo genere, perché AND ha precedenza su OR.

Autoverifica. Un brano Hard rock di 240 secondi compare?

Risposta. No. La traccia dice più lunghi di 240: il confronto corretto è >, non >=.

RICHIESTA 2 | QUERY COMMENTATA

# Ascolti completi di Data Science

Elencare nickname e titolo per ogni ascolto completo effettuato da uno studente di Data Science, ordinando per nickname e titolo.

Una riga = un evento di ascolto completo; non una coppia studente-brano distinta.

```
-- Una riga per ascolto completo: non aggiungere DISTINCT.
SELECT S.nickname, B.titolo
FROM STUDENTI S
-- Collega lo studente ai suoi eventi di ascolto.
JOIN ASCOLTI A ON A.studente = S.id_studente
-- Recupera titolo e durata del brano ascoltato.
JOIN BRANI B ON B.id_brano = A.brano
-- Considera soltanto gli studenti di Data Science.
WHERE S.corso_studio = 'Data Science'
-- Completo significa secondi ascoltati uguali alla durata.
  AND A.secondi_ascoltati = B.durata_secondi
-- Prima nickname, poi titolo all'interno dello stesso studente.
ORDER BY S.nickname, B.titolo;
```

Risultato verificato: 18 righe. Le ripetizioni previste dal PDF del docente devono restare presenti.

RICHIESTA 2 | IL RAGIONAMENTO

# Perché si scrive così

Leggi il percorso come una trasformazione del problema, non come una sequenza da memorizzare.

### 1. Individua la tabella degli eventi

ASCOLTI contiene chi ha ascoltato, quale brano e per quanti secondi. STUDENTI serve per nickname e corso; BRANI per titolo e durata totale.

### 2. Costruisci il percorso dei join

S.id_studente = A.studente collega la persona all'evento. B.id_brano = A.brano collega l'evento al brano corretto.

### 3. Definisci completo con precisione

La traccia richiede uguaglianza: A.secondi_ascoltati = B.durata_secondi. Non basta che i secondi ascoltati siano positivi.

### 4. Conserva gli eventi ripetuti

Due ascolti completi dello stesso brano, fatti dalla stessa persona, sono due eventi validi. ORDER BY ordina le righe ma non le unisce.

## Scelte e alternative

INNER JOIN per eventi esistenti. Cerchiamo ascolti effettivamente avvenuti: non serve conservare gli studenti senza ascolti. Un LEFT JOIN filtrato nel WHERE potrebbe comunque ridursi a un inner join, con maggiore confusione.

Niente DISTINCT. DISTINCT su nickname e titolo cancellerebbe un ascolto completo ripetuto. Sarebbe adatto a una traccia diversa: trovare le coppie studente-brano ascoltate almeno una volta per intero.

Uguaglianza invece di >=. Sui dati caricati nessun ascolto supera la durata. Tuttavia >= cambia la definizione della richiesta: qui seguiamo il significato esatto indicato dal docente.

> Esempio sui dati: Partenope ha due ascolti completi di I P' ME, TU P' TE. Le due righe uguali nel risultato sono corrette: la SELECT non mostra l'id dell'ascolto, ma gli eventi sottostanti restano diversi.

Errore da evitare. Usare GROUP BY nickname, titolo senza una necessità di aggregazione elimina la distinzione tra eventi, proprio come DISTINCT.

Autoverifica. Perché due righe identiche non provano che il JOIN sia sbagliato?

Risposta. Le colonne selezionate possono coincidere anche quando le righe di ASCOLTI hanno id_ascolto differenti.

RICHIESTA 3 | QUERY COMMENTATA

# Ascolti di aprile 2026

Mostrare studente, brano, artista e data/ora degli ascolti di aprile 2026, in ordine cronologico.

Una riga = un ascolto nel mese richiesto, con la sua data e ora completa.

```
-- Mantieni il dettaglio dell'evento e la data/ora completa.
SELECT S.nickname, B.titolo, AR.nome AS artista, A.data_ora
-- La data da filtrare appartiene ad ASCOLTI.
FROM ASCOLTI A
JOIN STUDENTI S ON S.id_studente = A.studente
JOIN BRANI B ON B.id_brano = A.brano
JOIN ARTISTI AR ON AR.id_artista = B.artista
-- Inizio di aprile incluso: nessuna conversione da stringa ambigua.
WHERE A.data_ora >= TIMESTAMP '2026-04-01 00:00:00'
-- Inizio di maggio escluso: comprende tutto il 30 aprile.
  AND A.data_ora < TIMESTAMP '2026-05-01 00:00:00'
-- Ordina sul timestamp, non sul suo testo formattato.
ORDER BY A.data_ora;
```

Risultato verificato: 12 ascolti. Il filtro mantiene anche la componente oraria e considera soltanto aprile 2026.

RICHIESTA 3 | IL RAGIONAMENTO

# Perché si scrive così

Leggi il percorso come una trasformazione del problema, non come una sequenza da memorizzare.

### 1. Parti da ASCOLTI

È qui che si trova A.data_ora. Gli altri tre join aggiungono le etichette leggibili senza trasformare gli eventi in un riepilogo.

### 2. Fissa un inizio incluso

TIMESTAMP '2026-04-01 00:00:00' indica l'inizio di aprile. >= include anche un ascolto esattamente a mezzanotte del primo giorno.

### 3. Fissa una fine esclusa

< TIMESTAMP '2026-05-01 00:00:00' esclude maggio ma comprende ogni istante di aprile, comprese eventuali frazioni di secondo dell'ultimo giorno.

### 4. Ordina per il valore temporale

ORDER BY A.data_ora usa il timestamp, non una sua rappresentazione testuale. L'orario non va scartato per mostrare solo il giorno.

## Scelte e alternative

Intervallo invece di EXTRACT. Il docente usa EXTRACT(YEAR)=2026 ed EXTRACT(MONTH)=4: è corretto e dà gli stessi risultati. L'intervallo rende i confini espliciti e non applica una funzione alla colonna; l'effettiva convenienza dipende da indici e piano di esecuzione.

Fine esclusa invece di BETWEEN. BETWEEN comprende entrambi gli estremi. Se l'estremo superiore fosse il 1 maggio, includerebbe anche quell'istante; se fosse il 30 aprile senza orario, escluderebbe gran parte dell'ultimo giorno.

Literal TIMESTAMP invece di stringhe ambigue. '01/04/2026' può dipendere dalle impostazioni NLS e dal formato atteso. Il literal TIMESTAMP esplicita tipo e formato, evitando questa ambiguità.

> Esempio sui dati: Il primo risultato è aurora, Another Brick in the Wall, Part 2, Pink Floyd, 1 aprile 2026 alle 16:46. L'ultimo è rocky, Thunderstruck, AC/DC, 23 aprile alle 14:04.

Errore da evitare. Filtrare soltanto il mese 4 includerebbe anche gli aprili di anni diversi se il dataset fosse ampliato.

Autoverifica. Un ascolto del 30 aprile alle 23:59:59.500 entra?

Risposta. Sì: è precedente al 1 maggio a mezzanotte. Non serve inventare l'ultimo istante rappresentabile del mese.

RICHIESTA 4 | QUERY COMMENTATA

# Numero e durata media per genere

Per ogni genere, contare i brani e calcolarne la durata media arrotondata a un decimale; ordinare per numerosità e poi per genere.

Una riga = un genere, non un singolo brano.

```
-- Una riga per genere; COUNT conta i brani del gruppo.
SELECT genere, COUNT(*) AS num_brani,
-- Arrotonda la media finale a un decimale.
       ROUND(AVG(durata_secondi), 1) AS durata_media
FROM BRANI
-- Forma un gruppo distinto per ciascun genere.
GROUP BY genere
-- Più brani prima; a parità, genere alfabetico.
ORDER BY num_brani DESC, genere;
```

Risultato verificato: 5 generi: Rock 6 brani; Hard rock, Pop, Rap e Rock progressivo 3 ciascuno.

RICHIESTA 4 | IL RAGIONAMENTO

# Perché si scrive così

Leggi il percorso come una trasformazione del problema, non come una sequenza da memorizzare.

### 1. Usa soltanto BRANI

Genere e durata sono già nella stessa tabella. Aggiungere ASCOLTI farebbe pesare di più i brani ascoltati molte volte.

### 2. Forma i gruppi

GROUP BY genere riunisce tutti i brani dello stesso genere. La colonna non aggregata selezionata è proprio il genere.

### 3. Calcola due misure diverse

COUNT(*) restituisce il numero dei brani del gruppo. AVG(durata_secondi) fa la media delle loro durate; ROUND(..., 1) arrotonda il numero finale.

### 4. Gestisci le parità nell'ordine

ORDER BY num_brani DESC, genere mette prima i gruppi più numerosi. A parità di conteggio, ordina alfabeticamente per genere.

## Scelte e alternative

COUNT(*) invece di COUNT(DISTINCT genere). Dentro ogni gruppo il genere è sempre lo stesso: COUNT(DISTINCT genere) varrebbe 1, non il numero dei brani.

AVG delle durate, non degli ascolti. La durata media dei brani descrive il catalogo. La media dei secondi_ascoltati descriverebbe un comportamento degli utenti, che è un'altra domanda.

Arrotondamento numerico invece di formato testuale. ROUND produce ancora un numero. Per mostrare sempre una cifra decimale si può aggiungere una formattazione TO_CHAR, ma il risultato diventerebbe testo. Il valore 318 e la visualizzazione 318,0 rappresentano la stessa media arrotondata.

> Esempio sui dati: Per il Rock progressivo: (334 + 238 + 382) / 3 = 318,0 secondi. I tre brani vengono contati una volta ciascuno, indipendentemente dagli ascolti ricevuti.

Errore da evitare. Scrivere SELECT genere, titolo, AVG(...) con GROUP BY genere non definisce quale titolo mostrare per il gruppo e in Oracle genera un errore di raggruppamento.

Autoverifica. Perché non c'è JOIN?

Risposta. Nessuna colonna richiesta si trova fuori da BRANI. Un join superfluo può cambiare la molteplicità delle righe.

RICHIESTA 5 | QUERY COMMENTATA

# Gruppi ascoltati da più di cinque studenti

Per i gruppi musicali ascoltati da più di cinque studenti distinti, mostrare nome, numero di studenti e secondi complessivi ascoltati.

Una riga = un artista di tipo GRUPPO che supera la soglia di studenti.

```
-- Mostra il nome del gruppo musicale.
SELECT AR.nome AS artista,
-- Ogni studente conta una sola volta per artista.
       COUNT(DISTINCT A.studente) AS num_studenti,
-- Tutti gli eventi contribuiscono ai secondi, anche se ripetuti.
       SUM(A.secondi_ascoltati) AS secondi_totali
FROM ARTISTI AR
JOIN BRANI B ON B.artista = AR.id_artista
JOIN ASCOLTI A ON A.brano = B.id_brano
-- Filtro sulle singole righe: escludi i solisti.
WHERE AR.tipo_artista = 'GRUPPO'
-- Una riga per artista; id e nome restano associati.
GROUP BY AR.id_artista, AR.nome
-- Filtro sui gruppi: più di cinque persone distinte.
HAVING COUNT(DISTINCT A.studente) > 5
-- Ordina per secondi totali, non per numero di studenti.
ORDER BY secondi_totali DESC;
```

Risultato verificato: Pink Floyd: 6 studenti, 4162 secondi. Dire Straits: 7, 4094. Queen: 8, 3347.

RICHIESTA 5 | IL RAGIONAMENTO

# Perché si scrive così

Leggi il percorso come una trasformazione del problema, non come una sequenza da memorizzare.

### 1. Collega artista, catalogo ed eventi

Il percorso ARTISTI -> BRANI -> ASCOLTI consente di ricondurre ogni evento al gruppo che ha prodotto il brano.

### 2. Filtra il tipo prima dei conteggi

WHERE AR.tipo_artista = 'GRUPPO' elimina i solisti. Questo criterio riguarda una proprietà della riga, non un risultato aggregato.

### 3. Conta le persone senza ripeterle

COUNT(DISTINCT A.studente) conta una persona una sola volta per artista, anche se ha ascoltato molti brani o ripetuto lo stesso brano.

### 4. Somma invece tutti gli eventi

SUM(A.secondi_ascoltati) comprende ogni ascolto. Dopo GROUP BY, HAVING verifica che il numero di persone distinte sia strettamente maggiore di cinque.

## Scelte e alternative

WHERE per il tipo; HAVING per la soglia. Il numero di studenti nasce solo dopo l'aggregazione. Mettere COUNT(...) nel WHERE dello stesso blocco non è la forma corretta per filtrare questi gruppi.

DISTINCT solo nel conteggio delle persone. SUM(DISTINCT secondi_ascoltati) sarebbe sbagliato: due ascolti diversi lunghi 200 secondi devono contribuire entrambi, non una sola volta.

Identificativo e nome nel GROUP BY. L'identificativo definisce l'artista; il nome è mostrato nel risultato. Nel dataset nome è anche UNIQUE: raggruppare soltanto per nome, come il docente, è qui equivalente.

> Esempio sui dati: Gli AC/DC hanno cinque studenti distinti: non entrano, perché > 5 non significa >= 5. I Pink Floyd ne hanno sei e totalizzano 4162 secondi.

Errore da evitare. COUNT(*) conta gli ascolti, non gli studenti. Un solo studente che ascolta dieci volte non soddisfa la richiesta di più di cinque persone.

Autoverifica. Uno studente ascolta tre brani dello stesso artista: quanto pesa nelle due misure?

Risposta. Pesa uno nel conteggio degli studenti distinti; i secondi di tutti e tre gli ascolti entrano nella somma.

RICHIESTA 6 | QUERY COMMENTATA

# Percentuale media di ascolto sotto il 90%

Calcolare per ogni brano la percentuale media ascoltata e mostrare soltanto le medie inferiori al 90%, con un decimale.

Una riga = un brano con almeno un ascolto e con media dei rapporti inferiore alla soglia.

```
-- Il risultato resta al livello del singolo brano.
SELECT B.titolo,
-- Media dei rapporti, poi percentuale e arrotondamento.
       ROUND(AVG(A.secondi_ascoltati / B.durata_secondi) * 100, 1) AS perc_media
FROM BRANI B
-- I brani senza eventi non hanno una media osservata.
JOIN ASCOLTI A ON A.brano = B.id_brano
-- L'id impedisce di fondere eventuali brani con lo stesso titolo.
GROUP BY B.id_brano, B.titolo
-- La soglia si applica alla media esatta, prima di ROUND.
HAVING AVG(A.secondi_ascoltati / B.durata_secondi) < 0.9
-- Percentuali mostrate crescenti; titolo in caso di parità.
ORDER BY perc_media, B.titolo;
```

Risultato verificato: 5 brani: Back in Black 82,4%; Non dirgli mai 83,3%; Another Brick in the Wall, Part 2 85,0%; Bohemian Rhapsody 86,6%; COME VUOI 87,0%.

RICHIESTA 6 | IL RAGIONAMENTO

# Perché si scrive così

Leggi il percorso come una trasformazione del problema, non come una sequenza da memorizzare.

### 1. Calcola un rapporto per evento

A.secondi_ascoltati / B.durata_secondi esprime la frazione di brano ascoltata in quell'evento. Ad esempio 120/255 è circa 0,4706.

### 2. Fai la media nello stesso brano

GROUP BY B.id_brano, B.titolo separa i brani anche se un futuro catalogo contenesse titoli uguali. AVG attribuisce un peso uguale a ciascun ascolto.

### 3. Trasforma e arrotonda

Moltiplicare per 100 converte la frazione in percentuale. ROUND(..., 1) decide soltanto come arrotondare il valore mostrato.

### 4. Filtra il valore esatto

HAVING confronta la media non arrotondata con 0.9. Il confronto prima dell'arrotondamento evita di alterare il significato di inferiore al 90%.

## Scelte e alternative

AVG dei rapporti invece della sola somma. SUM(secondi_ascoltati)/durata supera anche il 100% quando il brano viene ascoltato più volte. Non è una percentuale media per evento.

INNER JOIN invece di inventare uno zero. Un brano mai ascoltato non ha una media osservata. Escluderlo segue il docente. Trattarlo come 0% con un LEFT JOIN e COALESCE sarebbe una scelta di significato diversa, da dichiarare.

Soglia non arrotondata. Una media del 89,96% soddisfa la condizione < 90 anche se, arrotondata a un decimale, si visualizza 90,0. Filtrare il valore visualizzato cambierebbe il risultato al bordo della soglia.

> Esempio sui dati: Back in Black ha tre ascolti di 255, 255 e 120 secondi, su una durata di 255: ((255/255 + 255/255 + 120/255) / 3) x 100 = 82,3529...; il risultato mostrato è 82,4%.

Errore da evitare. Fare prima la media di tutti gli ascolti del database e poi dividerla per la durata di un brano mescola gruppi e unità diverse.

Autoverifica. DIO LO SA, che non ha ascolti, deve comparire come 0%?

Risposta. No in questa soluzione: la media non è definita in assenza di eventi. La richiesta 7 gestisce invece esplicitamente i brani mai ascoltati.

RICHIESTA 7 | QUERY COMMENTATA

# Tutti i brani, anche senza ascolti

Contare gli ascolti ricevuti da ogni brano, mostrando anche i brani mai ascoltati con conteggio zero.

Una riga = un brano del catalogo, anche se non compare in ASCOLTI.

```
-- Conta gli id degli eventi: COUNT(*) darebbe 1 ai mai ascoltati.
SELECT B.titolo, COUNT(A.id_ascolto) AS num_ascolti
-- Tutti i brani sono l'insieme da conservare.
FROM BRANI B
-- Conserva anche i brani senza righe in ASCOLTI.
LEFT JOIN ASCOLTI A ON A.brano = B.id_brano
-- Un gruppo per brano, identificato dalla sua chiave.
GROUP BY B.id_brano, B.titolo
-- Prima i meno ascoltati; poi il titolo per le parità.
ORDER BY num_ascolti, B.titolo;
```

Risultato verificato: 18 righe. DIO LO SA ha 0 ascolti; I P' ME, TU P' TE ne ha 9 ed è l'ultimo nell'ordine crescente.

RICHIESTA 7 | IL RAGIONAMENTO

# Perché si scrive così

Leggi il percorso come una trasformazione del problema, non come una sequenza da memorizzare.

### 1. Metti BRANI a sinistra

È l'insieme che vuoi conservare integralmente. LEFT JOIN tiene tutti i brani e affianca gli eventuali eventi corrispondenti.

### 2. Riconosci la riga senza corrispondenza

Per un brano senza ascolti, il risultato del join contiene comunque una riga del brano; le colonne di A sono NULL.

### 3. Conta una colonna non nulla degli eventi

A.id_ascolto è una chiave primaria. COUNT(A.id_ascolto) ignora il NULL prodotto dal mancato abbinamento e restituisce zero.

### 4. Raggruppa e ordina

Il raggruppamento per id e titolo restituisce una riga per brano. L'ordine usa prima il conteggio crescente e poi il titolo per le parità.

## Scelte e alternative

LEFT JOIN invece di INNER JOIN. L'inner join escluderebbe DIO LO SA perché non trova eventi. Non può quindi soddisfare la parte della traccia che impone la presenza dei mai ascoltati.

COUNT(id_ascolto) invece di COUNT(*). COUNT(*) conterebbe anche l'unica riga conservata dal left join e darebbe 1 al brano senza eventi. COUNT della chiave dell'evento dà il valore corretto 0.

Niente COALESCE sul conteggio. COUNT restituisce già un numero, anche quando è zero. COALESCE è utile per altre aggregazioni che possono restituire NULL, ma qui è superfluo.

> Esempio sui dati: DIO LO SA resta nel risultato con A.id_ascolto nullo dopo il join. COUNT(A.id_ascolto)=0, mentre COUNT(*)=1: è il caso che distingue una soluzione corretta da una quasi corretta.

Errore da evitare. Un filtro su una colonna di A nel WHERE può cancellare proprio la riga senza abbinamento, vanificando il left join.

Autoverifica. Due brani con lo stesso titolo verrebbero fusi?

Risposta. No: il GROUP BY include anche B.id_brano. Il titolo serve a mostrarli e ordinarli, non a identificarli da solo.

RICHIESTA 8 | QUERY COMMENTATA

# Studenti e ascolti per corso

Per ogni corso, contare gli studenti iscritti e gli ascolti complessivi, includendo i corsi con zero ascolti.

Una riga = un corso presente in STUDENTI.

```
-- Riepilogo al livello del corso.
SELECT S.corso_studio,
-- Evita di ricontare la persona a ogni suo ascolto.
       COUNT(DISTINCT S.id_studente) AS num_studenti,
-- Conta gli eventi reali; nessun evento significa zero.
       COUNT(A.id_ascolto) AS num_ascolti
FROM STUDENTI S
-- Mantieni anche gli iscritti senza ascolti.
LEFT JOIN ASCOLTI A ON A.studente = S.id_studente
-- Unisci i conteggi degli studenti dello stesso corso.
GROUP BY S.corso_studio
-- Più ascolti prima; il corso risolve eventuali parità.
ORDER BY num_ascolti DESC, S.corso_studio;
```

Risultato verificato: Informatica 4/33; Data Science 3/20; Ingegneria informatica 3/19; Matematica 1/0 (iscritti/ascolti).

RICHIESTA 8 | IL RAGIONAMENTO

# Perché si scrive così

Leggi il percorso come una trasformazione del problema, non come una sequenza da memorizzare.

### 1. Parti dagli iscritti

La tabella STUDENTI contiene il corso e le persone. LEFT JOIN consente di tenere anche uno studente senza alcun ascolto.

### 2. Osserva la moltiplicazione delle righe

Dopo il join, uno studente con nove ascolti compare in nove righe. Quelle righe servono per contare gli eventi, ma non sono nove persone.

### 3. Usa due conteggi con significati diversi

COUNT(DISTINCT S.id_studente) conta gli iscritti una volta sola. COUNT(A.id_ascolto) conta invece tutti gli eventi, ignorando gli abbinamenti assenti.

### 4. Aggrega al livello del corso

GROUP BY S.corso_studio produce il riepilogo. Il primo criterio di ordinamento è il numero di ascolti decrescente.

## Scelte e alternative

DISTINCT sugli studenti, non sugli ascolti. Ogni ascolto ha già un id univoco: non è necessario un DISTINCT per contarlo. Senza DISTINCT sullo studente, invece, gli utenti più attivi gonfierebbero il numero degli iscritti.

Conservare i corsi senza eventi. Un inner join perderebbe Matematica, perché il suo unico studente non ascolta nulla. Il left join mantiene quel corso con un iscritto e zero eventi.

Limite dello schema disponibile. Non esiste una tabella CORSI. Possiamo mostrare i corsi che hanno almeno uno studente; un corso senza alcun iscritto non è rappresentato nei dati e non può essere inventato dalla query.

> Esempio sui dati: Informatica ha 4 studenti e 33 ascolti. Contare semplicemente le righe del join confonderebbe questi due numeri. Matematica ha invece 1 studente e 0 ascolti.

Errore da evitare. COUNT(S.id_studente) dopo il join non è il numero degli iscritti: ogni evento ripete lo stesso id.

Autoverifica. Se un corso ha tre studenti e tutti ascoltano dieci volte, quali numeri vuoi ottenere?

Risposta. Tre iscritti e trenta ascolti. DISTINCT opera soltanto sulla prima misura.

RICHIESTA 9 | QUERY COMMENTATA

# Brano più lungo

Trovare titolo, artista e durata del brano più lungo, conservando gli eventuali pari merito.

Una riga = un brano la cui durata è uguale al massimo globale.

```
-- Il titolo e l'artista devono appartenere alla riga del massimo.
SELECT B.titolo, AR.nome AS artista, B.durata_secondi
FROM BRANI B
JOIN ARTISTI AR ON AR.id_artista = B.artista
-- Conserva tutti gli eventuali brani a pari durata massima.
WHERE B.durata_secondi = (SELECT MAX(durata_secondi) FROM BRANI)
-- Ordina alfabeticamente eventuali pari merito.
ORDER BY B.titolo;
```

Risultato verificato: Una riga: Money for Nothing, Dire Straits, 505 secondi.

RICHIESTA 9 | IL RAGIONAMENTO

# Perché si scrive così

Leggi il percorso come una trasformazione del problema, non come una sequenza da memorizzare.

### 1. Calcola il massimo come valore

La sottoquery SELECT MAX(durata_secondi) FROM BRANI restituisce un solo numero: la maggiore durata presente nel catalogo.

### 2. Confronta ogni brano con quel numero

Il WHERE conserva tutte le righe che hanno esattamente quella durata, non soltanto la prima trovata.

### 3. Aggiungi il nome dell'artista

Il join con ARTISTI serve a mostrare il nome, che non è memorizzato direttamente in BRANI.

### 4. Definisci un ordine per eventuali parità

ORDER BY B.titolo rende leggibile e stabile il risultato in presenza di più brani con durata massima.

## Scelte e alternative

MAX con uguaglianza invece di tagliare a una riga. ORDER BY durata DESC FETCH FIRST 1 ROW ONLY perderebbe altri brani eventualmente a pari merito. In Oracle FETCH FIRST 1 ROW WITH TIES è un'alternativa valida se l'ordinamento considera la sola durata.

Non basta selezionare MAX e titolo insieme. Il massimo è un valore aggregato; il titolo appartiene a una riga. SELECT titolo, MAX(durata) senza una logica di associazione non identifica automaticamente il titolo corretto.

Non serve ordinare la sottoquery. MAX trova già il valore maggiore. La sottoquery non deve restituire un elenco ordinato né tutte le durate.

> Esempio sui dati: La durata massima è 505 secondi. La riga corrispondente è Money for Nothing dei Dire Straits. Se un secondo brano durasse 505 secondi, verrebbe mostrato anch'esso.

Errore da evitare. Aggiungere anche il titolo all'ORDER BY di un FETCH ... WITH TIES cambia il concetto di parità: due titoli diversi non sarebbero più pari su tutte le chiavi d'ordine.

Autoverifica. Se la tabella fosse vuota, apparirebbe una riga con titolo nullo?

Risposta. No. MAX restituirebbe NULL, ma non ci sarebbe alcuna riga di BRANI da selezionare e il confronto con NULL non risulterebbe vero.

RICHIESTA 10 | QUERY COMMENTATA

# Brano con più ascolti

Trovare il brano con il maggior numero di ascolti, mostrando anche il conteggio e gli eventuali pari merito.

Una riga = un brano che raggiunge il massimo dei conteggi degli ascolti.

```
-- COUNT(*) è corretto qui: l'INNER JOIN contiene solo eventi reali.
SELECT B.titolo, COUNT(*) AS num_ascolti
FROM BRANI B
JOIN ASCOLTI A ON A.brano = B.id_brano
-- Conta gli ascolti separatamente per ogni brano.
GROUP BY B.id_brano, B.titolo
-- Confronta il conteggio del gruppo con il massimo dei conteggi.
HAVING COUNT(*) = (
-- La query intermedia riduce tutti i conteggi a un solo massimo.
    SELECT MAX(num_ascolti)
-- La query interna calcola prima un conteggio per brano.
    FROM (SELECT COUNT(*) AS num_ascolti FROM ASCOLTI GROUP BY brano)
)
-- Non tagliare a una sola riga: potrebbero esserci pari merito.
ORDER BY B.titolo;
```

Risultato verificato: Una riga: I P' ME, TU P' TE, 9 ascolti.

RICHIESTA 10 | IL RAGIONAMENTO

# Perché si scrive così

Leggi il percorso come una trasformazione del problema, non come una sequenza da memorizzare.

### 1. Conta gli eventi per brano

Il GROUP BY esterno forma un gruppo per B.id_brano e titolo. Poiché si usa un inner join, ciascuna riga del gruppo è un ascolto reale e COUNT(*) è appropriato.

### 2. Crea l'elenco dei conteggi

La sottoquery più interna raggruppa ASCOLTI per brano e produce un numero di eventi per ciascun brano ascoltato.

### 3. Trova il massimo tra i conteggi

SELECT MAX(num_ascolti) legge quell'elenco derivato e restituisce il numero di ascolti più alto.

### 4. Confronta i gruppi con il massimo

HAVING COUNT(*) = (...) conserva tutti i brani che raggiungono il massimo, anche quando sono più di uno.

## Scelte e alternative

Conteggio degli eventi invece degli utenti. COUNT(DISTINCT A.studente) risponderebbe a quale brano è stato ascoltato dal maggior numero di persone. La traccia chiede invece quanti ascolti ha ricevuto.

Nessun limite arbitrario a una riga. LIMIT non è la sintassi Oracle. Anche FETCH FIRST 1 ROW ONLY potrebbe perdere i pari merito. Il confronto con il massimo li conserva.

Alternativa con CTE o funzione analitica. Si può prima costruire una CTE dei conteggi e poi confrontarli con MAX, oppure applicare DENSE_RANK sui conteggi. Sono alternative valide; qui la struttura segue quella del docente e mantiene visibili le due aggregazioni.

> Esempio sui dati: I P' ME, TU P' TE riceve nove ascolti. Il massimo dei conteggi è quindi 9 e soltanto quel brano lo raggiunge nel dataset originale.

Errore da evitare. Questa soluzione considera i brani con almeno un ascolto. Se ASCOLTI fosse completamente vuota, non restituirebbe i brani tutti a zero: servirebbe una diversa definizione basata su LEFT JOIN.

Autoverifica. Lo stesso studente ascolta il brano cinque volte: quanto aggiunge al conteggio?

Risposta. Cinque. Si contano eventi, non persone distinte.

RICHIESTA 11 | QUERY COMMENTATA

# Studenti con più ascolti

Trovare tutti gli studenti con il numero massimo di ascolti, mostrando nickname e conteggio.

Una riga = uno studente che raggiunge il massimo degli ascolti effettuati.

```
-- Ogni evento pesa uno, anche se ripete lo stesso brano.
SELECT S.nickname, COUNT(*) AS num_ascolti
FROM STUDENTI S
JOIN ASCOLTI A ON A.studente = S.id_studente
-- Una riga aggregata per studente, non per corso.
GROUP BY S.id_studente, S.nickname
-- Seleziona ogni studente che raggiunge il massimo.
HAVING COUNT(*) = (
-- Trova il più alto tra i conteggi per persona.
    SELECT MAX(num_ascolti)
-- Prima conta gli eventi di ciascuno studente.
    FROM (SELECT COUNT(*) AS num_ascolti FROM ASCOLTI GROUP BY studente)
)
-- Mantieni tutti i pari merito e ordinali per nickname.
ORDER BY S.nickname;
```

Risultato verificato: Due righe: ada.exe 9; bytebeat 9.

RICHIESTA 11 | IL RAGIONAMENTO

# Perché si scrive così

Leggi il percorso come una trasformazione del problema, non come una sequenza da memorizzare.

### 1. Associa eventi e studenti

ASCOLTI fornisce gli eventi; STUDENTI permette di tradurre l'id in nickname. Ogni evento è contato una volta.

### 2. Raggruppa per persona

GROUP BY id_studente e nickname impedisce di confondere lo studente con il brano o con il suo corso di studi.

### 3. Calcola separatamente il massimo

La sottoquery conta gli ascolti di ogni studente e MAX seleziona il maggiore tra questi conteggi.

### 4. Mantieni tutte le persone a pari merito

HAVING confronta ogni conteggio col massimo. ORDER BY nickname ordina i vincitori alfabeticamente, senza ridurne il numero.

## Scelte e alternative

Niente conteggio distinto dei brani. COUNT(DISTINCT A.brano) misurerebbe quanti titoli diversi lo studente ha ascoltato. Gli ascolti ripetuti dello stesso brano devono contare nella richiesta attuale.

Non scegliere una sola riga. I dati contengono un pari merito reale: ada.exe e bytebeat hanno nove ascolti ciascuno. Tagliare alla prima riga produrrebbe una risposta incompleta.

Inner join coerente con il docente. Su questi dati il massimo è positivo, quindi gli studenti senza eventi non possono vincere. Con una tabella ASCOLTI vuota, una formulazione con LEFT JOIN potrebbe invece trattare tutti gli studenti a quota zero: sarebbe un'estensione della convenzione adottata qui.

> Esempio sui dati: Il massimo è 9. ada.exe e bytebeat hanno entrambi quel valore e compaiono entrambi. Silenzio non compare perché non ha effettuato alcun ascolto.

Errore da evitare. Raggruppare per corso anziché per studente risponderebbe a una domanda diversa, già affrontata nella richiesta 8.

Autoverifica. Perché il massimo non si calcola con MAX(id_ascolto)?

Risposta. L'id identifica un evento e non rappresenta il numero di eventi di una persona. Occorre prima contare per studente.

RICHIESTA 12 | QUERY COMMENTATA

# Studenti che hanno ascoltato artisti italiani

Elencare i nickname degli studenti che hanno ascoltato almeno un brano di un artista italiano, senza ripetizioni.

Una riga = uno studente che soddisfa almeno una volta la condizione.

```
-- La query esterna elenca persone, non eventi.
SELECT S.nickname
FROM STUDENTI S
-- Basta almeno un ascolto italiano per ammettere lo studente.
WHERE EXISTS (
-- Il valore 1 è convenzionale: interessa solo l'esistenza.
    SELECT 1
    FROM ASCOLTI A
    JOIN BRANI B ON B.id_brano = A.brano
    JOIN ARTISTI AR ON AR.id_artista = B.artista
-- Correla l'evento alla persona esterna e verifica il paese.
    WHERE A.studente = S.id_studente AND AR.paese = 'Italia'
)
-- Ogni nickname compare una sola volta, in ordine alfabetico.
ORDER BY S.nickname;
```

Risultato verificato: 8 nickname: ada.exe, aurora, bytebeat, clef, echo, luna.sql, partenope, synthwave.

RICHIESTA 12 | IL RAGIONAMENTO

# Perché si scrive così

Leggi il percorso come una trasformazione del problema, non come una sequenza da memorizzare.

### 1. Parti dalle persone da elencare

La SELECT esterna legge STUDENTI: ogni sua riga può entrare una volta, indipendentemente da quanti ascolti italiani ha effettuato.

### 2. Definisci un ascolto che soddisfa la richiesta

Nella sottoquery, ASCOLTI si collega a BRANI e ARTISTI; AR.paese deve essere Italia.

### 3. Correla la sottoquery alla persona

A.studente = S.id_studente è il legame decisivo: chiede se esiste un evento valido di quello specifico studente.

### 4. Usa EXISTS come domanda sì/no

Se esiste almeno una riga, lo studente entra. Il numero di righe trovate non aumenta il numero di volte in cui appare il nickname.

## Scelte e alternative

EXISTS invece di un join che moltiplica le persone. Un JOIN nella query esterna con SELECT DISTINCT nickname sarebbe anch'esso valido. EXISTS rende più diretto il significato almeno uno, senza creare prima le ripetizioni nella forma logica della query.

Alternativa IN del docente. id_studente IN (sottoquery degli id che hanno ascoltato artisti italiani) è equivalente in questo caso. La scelta di EXISTS è di chiarezza semantica, non una garanzia di maggiore velocità.

SELECT 1 senza promesse di prestazioni. Dentro EXISTS interessa che una riga esista, non quali valori mostri. SELECT 1 e SELECT * sono entrambi corretti; 1 rende esplicita l'intenzione, non dimostra un vantaggio prestazionale.

> Esempio sui dati: Partenope ascolta più volte artisti italiani, ma il nickname appare una sola volta. Gli artisti italiani presenti sono Gigi D'Alessio e Geolier.

Errore da evitare. Se dimentichi A.studente = S.id_studente, basta un solo ascolto italiano fatto da chiunque per rendere EXISTS vero per tutti gli studenti.

Autoverifica. Che cosa cambierebbe chiedendo il numero degli ascolti italiani per ogni studente?

Risposta. Servirebbe una vera aggregazione per persona con COUNT. EXISTS da solo risponde soltanto alla presenza o assenza di almeno un evento.

RICHIESTA 13 | QUERY COMMENTATA

# Brani sopra la media del proprio genere

Trovare i brani più lunghi della media delle durate dei brani appartenenti allo stesso genere.

Una riga = un brano sopra la media del proprio gruppo di genere.

```
-- Restituisci i dati del brano esterno B.
SELECT B.titolo, B.genere, B.durata_secondi
FROM BRANI B
-- Confronta la sua durata con una media numerica.
WHERE B.durata_secondi > (
-- La media riguarda i brani, non gli ascolti.
    SELECT AVG(B2.durata_secondi)
    FROM BRANI B2
-- Correlazione: considera solo il genere del brano B.
    WHERE B2.genere = B.genere
)
-- Generi alfabetici; dentro il genere, durate decrescenti.
ORDER BY B.genere, B.durata_secondi DESC;
```

Risultato verificato: 9 brani. Per Rock progressivo compaiono Comfortably Numb e Wish You Were Here; per Hard rock Thunderstruck e Back in Black.

RICHIESTA 13 | IL RAGIONAMENTO

# Perché si scrive così

Leggi il percorso come una trasformazione del problema, non come una sequenza da memorizzare.

### 1. Mantieni il brano esterno

B è il brano che stai valutando. Titolo, genere e durata mostrati provengono da quella riga.

### 2. Costruisci il gruppo di confronto

B2 è una seconda lettura di BRANI. Il filtro B2.genere = B.genere limita il calcolo ai brani con lo stesso genere del brano esterno.

### 3. Calcola e confronta

AVG(B2.durata_secondi) restituisce la media del genere. Il confronto > conserva soltanto i brani strettamente sopra quella media.

### 4. Ordina in due livelli

Prima i generi in ordine alfabetico; dentro ogni genere, le durate dalla più alta alla più bassa.

## Scelte e alternative

Media del genere invece di media globale. Togliendo la correlazione sul genere si confronterebbero tutti i brani con una sola media del catalogo, rispondendo a una domanda diversa.

Niente arrotondamento nella soglia. La traccia non chiede di arrotondare la media di confronto. Farlo potrebbe cambiare quali durate sono considerate sopra la media nei casi al limite.

Alternative equivalenti. Una CTE con una media per genere, poi un JOIN, è corretta. Anche AVG(...) OVER (PARTITION BY genere) è possibile dentro una sottoquery. Qui la correlazione mostra direttamente che ogni brano usa il proprio genere.

> Esempio sui dati: Nel Rock progressivo la media è 318 secondi. Comfortably Numb (382) e Wish You Were Here (334) la superano; Another Brick in the Wall, Part 2 (238) no.

Errore da evitare. Eliminare il brano corrente dalla sottoquery cambierebbe il gruppo su cui è definita la media richiesta. La traccia considera tutti i brani dello stesso genere, incluso il brano valutato.

Autoverifica. Se un genere contenesse un solo brano, quel brano entrerebbe?

Risposta. No: la sua durata sarebbe uguale alla media del genere, non maggiore.

RICHIESTA 14 | QUERY COMMENTATA

# Studenti che non hanno mai ascoltato Rap

Elencare gli studenti che non hanno mai ascoltato un brano Rap, includendo chi non ha ascoltato nulla.

Una riga = uno studente per cui non esiste alcun ascolto Rap.

```
-- Partire da STUDENTI conserva anche chi non ha eventi.
SELECT S.nickname
FROM STUDENTI S
-- Non deve esistere alcun ascolto Rap di questa persona.
WHERE NOT EXISTS (
    SELECT 1
    FROM ASCOLTI A
    JOIN BRANI B ON B.id_brano = A.brano
-- Entrambe le condizioni definiscono l'evento vietato.
    WHERE A.studente = S.id_studente AND B.genere = 'Rap'
)
-- Nickname alfabetici; nessun DISTINCT necessario.
ORDER BY S.nickname;
```

Risultato verificato: 4 nickname: clef, floydfan, rocky, silenzio.

RICHIESTA 14 | IL RAGIONAMENTO

# Perché si scrive così

Leggi il percorso come una trasformazione del problema, non come una sequenza da memorizzare.

### 1. Definisci prima l'evento vietato

Un ascolto Rap è una riga di ASCOLTI collegata a un brano con B.genere = 'Rap'.

### 2. Collegalo allo studente esterno

A.studente = S.id_studente impedisce che gli ascolti di un'altra persona influenzino la decisione.

### 3. Nega l'esistenza di quell'evento

NOT EXISTS è vero quando la sottoquery non trova nessun ascolto Rap di quella persona.

### 4. Conserva anche l'assenza totale di eventi

Per silenzio la sottoquery non trova alcun evento, quindi nemmeno un evento Rap: lo studente soddisfa correttamente la condizione.

## Scelte e alternative

NOT EXISTS invece di genere <> 'Rap'. Aver ascoltato un brano non Rap non esclude che lo stesso studente abbia ascoltato anche Rap. Un filtro <> su singole righe non dimostra mai ascoltato Rap.

NOT IN come alternativa condizionata. Un NOT IN sugli id degli ascoltatori Rap potrebbe funzionare qui perché ASCOLTI.studente è NOT NULL. In generale, però, un NULL nella sottoquery complica la logica di NOT IN. NOT EXISTS esplicita la mancanza di eventi correlati.

LEFT JOIN come altra possibilità. Un anti-join con LEFT JOIN e controllo IS NULL è valido se costruito contro il solo insieme degli ascolti Rap. La forma NOT EXISTS rende più semplice evitare filtri esterni che cancellino studenti senza eventi.

> Esempio sui dati: Clef, floydfan e rocky hanno ascolti, ma nessuno Rap. Silenzio non ne ha nessuno: tutti e quattro devono comparire.

Errore da evitare. Partire da STUDENTI JOIN ASCOLTI con un inner join elimina silenzio prima ancora di verificare la condizione mai Rap.

Autoverifica. Uno studente ascolta un Rock e un Rap: deve entrare?

Risposta. No. È sufficiente un solo ascolto Rap per rendere falsa la condizione non ha mai ascoltato Rap.

RICHIESTA 15 | QUERY COMMENTATA

# Studenti che hanno ascoltato tutti i brani dei Pink Floyd

Trovare gli studenti che hanno ascoltato tutti i brani dei Pink Floyd presenti nel catalogo.

Una riga = uno studente al quale non manca nessun brano dell'insieme richiesto.

```
-- Valuta una persona alla volta.
SELECT S.nickname
FROM STUDENTI S
-- Accetta se non esiste alcun brano richiesto che manca.
WHERE NOT EXISTS (
-- Questo blocco cerca i brani mancanti dei Pink Floyd.
    SELECT 1
    FROM BRANI B
    JOIN ARTISTI AR ON AR.id_artista = B.artista
-- Definisci l'insieme completo dei brani da aver ascoltato.
    WHERE AR.nome = 'Pink Floyd'
-- Un brano manca se non esiste un suo ascolto dello studente.
      AND NOT EXISTS (
          SELECT 1 FROM ASCOLTI A
-- Correla contemporaneamente persona e brano.
          WHERE A.studente = S.id_studente AND A.brano = B.id_brano
      )
)
-- Restituisci i nickname che hanno coperto tutto l'insieme.
ORDER BY S.nickname;
```

Risultato verificato: 2 nickname: aurora, floydfan. Caso limite dichiarato: senza brani dei Pink Floyd, questa forma selezionerebbe tutti gli studenti.

RICHIESTA 15 | IL RAGIONAMENTO

# Perché si scrive così

Leggi il percorso come una trasformazione del problema, non come una sequenza da memorizzare.

### 1. Definisci l'insieme da coprire

La prima sottoquery seleziona i brani il cui artista è Pink Floyd. Nel dataset sono tre brani distinti.

### 2. Cerca un ascolto del singolo brano

La sottoquery più interna verifica insieme la persona (A.studente = S.id_studente) e il brano corrente (A.brano = B.id_brano).

### 3. Trova gli eventuali brani mancanti

Il NOT EXISTS interno è vero per un brano che quello studente non ha mai ascoltato. La sottoquery intermedia diventa quindi l'elenco dei suoi brani mancanti.

### 4. Richiedi che non ne manchi nessuno

Il NOT EXISTS esterno accetta lo studente solo quando l'elenco dei mancanti è vuoto. È la riscrittura logica di ha ascoltato tutti.

## Scelte e alternative

Doppia negazione invece di un singolo EXISTS. Un solo EXISTS dimostrerebbe almeno un brano dei Pink Floyd. La parola tutti richiede che ogni brano dell'insieme sia coperto.

Ascolti ripetuti senza effetti. Esistere una volta o più volte è sufficiente per ciascun brano. Contare gli eventi con COUNT(*) e confrontarli con 3 sarebbe sbagliato: tre ripetizioni dello stesso titolo non coprono tre titoli diversi.

Alternativa con COUNT(DISTINCT brano). Si può contare quanti brani distinti dei Pink Floyd ha ascoltato ogni studente e confrontare con il numero dei brani dell'artista. Va gestito consapevolmente il caso di catalogo vuoto: il doppio NOT EXISTS considera soddisfatto tutti su un insieme vuoto.

> Esempio sui dati: Aurora e floydfan hanno ascoltato tutti e tre i brani. Non occorre che ogni ascolto sia completo: la traccia richiede l'esistenza dell'ascolto, non l'uguaglianza con la durata.

Errore da evitare. Dimenticare la correlazione A.brano = B.id_brano trasforma la domanda: basterebbe un ascolto qualsiasi della persona per far sembrare coperti tutti i brani.

Autoverifica. Se un quarto brano dei Pink Floyd fosse aggiunto e nessuno l'avesse ascoltato?

Risposta. Nessuno degli studenti attuali soddisferebbe più la richiesta. La query adatta automaticamente l'insieme da coprire.

RICHIESTA 16 | QUERY COMMENTATA

# Vista di dettaglio e riepilogo Data Science

Creare una vista con il dettaglio degli ascolti e usarla per riepilogare per artista gli ascolti di Data Science e i secondi totali.

Nella vista: un ascolto. Nella query finale: un artista ascoltato da studenti di Data Science.

```
-- Prima istruzione: definisci una vista ordinaria, non una copia dei dati.
CREATE OR REPLACE VIEW V_ASCOLTI_DETTAGLIO AS
-- Una riga della vista corrisponde a un singolo ascolto.
SELECT S.nickname, S.corso_studio, B.titolo, B.genere,
       AR.nome AS artista, A.data_ora, A.secondi_ascoltati
FROM ASCOLTI A
JOIN STUDENTI S ON S.id_studente = A.studente
JOIN BRANI B ON B.id_brano = A.brano
JOIN ARTISTI AR ON AR.id_artista = B.artista;

-- Seconda istruzione: usa la vista per il riepilogo richiesto.
-- Una riga finale per artista: conta eventi e somma secondi.
SELECT artista, COUNT(*) AS num_ascolti,
       SUM(secondi_ascoltati) AS secondi_totali
-- Riusa la vista creata dall'istruzione precedente.
FROM V_ASCOLTI_DETTAGLIO
-- Filtro sul corso prima delle aggregazioni.
WHERE corso_studio = 'Data Science'
-- Raggruppa il dettaglio della vista per artista.
GROUP BY artista
-- Prima il numero di ascolti; a parità, nome alfabetico.
ORDER BY num_ascolti DESC, artista;
```

Risultato verificato: 6 artisti. Primo Geolier 7/1264; poi Gigi D'Alessio 5/1108; Queen 3/685; AC/DC 2/412; Dire Straits 2/597; Pink Floyd 1/382 (ascolti/secondi).

RICHIESTA 16 | IL RAGIONAMENTO

# Perché si scrive così

Leggi il percorso come una trasformazione del problema, non come una sequenza da memorizzare.

### 1. Costruisci il dettaglio riutilizzabile

I quattro join ricompongono nickname, corso, brano, genere, artista, timestamp e secondi del singolo evento.

### 2. Dai un nome non ambiguo alla colonna

AR.nome AS artista definisce il nome della colonna della vista. La query successiva può quindi riferirsi direttamente ad artista.

### 3. Crea prima la vista

CREATE OR REPLACE VIEW memorizza la definizione della query. Non crea una copia autonoma dei 72 ascolti: una vista ordinaria legge le tabelle sottostanti.

### 4. Interroga la vista al livello richiesto

Il WHERE sceglie Data Science; GROUP BY artista forma un gruppo per artista. COUNT(*) conta gli ascolti e SUM somma i secondi; l'ordine principale usa il conteggio.

## Scelte e alternative

Vista perché richiesta e riutilizzabile. Una CTE o la ripetizione dei join dentro la SELECT finale potrebbe dare gli stessi numeri, ma non soddisfarebbe la parte esplicita creare una vista nominata V_ASCOLTI_DETTAGLIO.

COUNT(*) nella vista di dettaglio. La vista ha una riga per ascolto reale, senza righe artificiali di un outer join. COUNT(*) è quindi il conteggio appropriato degli eventi.

Numero di ascolti come criterio principale. Dal più ascoltato è interpretato come numero di eventi, coerentemente al docente. Ordinare per secondi_totali darebbe un criterio diverso: quantità di tempo ascoltato.

> Esempio sui dati: Geolier ha 7 ascolti e 1264 secondi. AC/DC e Dire Straits hanno entrambi 2 ascolti: AC/DC precede Dire Straits per il secondo criterio alfabetico, anche se ha meno secondi.

Errore da evitare. Una vista non conserva l'ordine di presentazione come proprietà garantita. L'ORDER BY va nella SELECT che produce il risultato da leggere.

Autoverifica. Il comando CREATE VIEW si esegue sulla connessione SYSTEM?

Risposta. Qui si esegue su UNINA, che possiede le tabelle e ha già il privilegio CREATE VIEW. Il file contiene il DDL prima della SELECT che lo usa.

RICHIESTA 17a | QUERY COMMENTATA

# Pink Floyd ma non Queen

Trovare gli studenti che hanno ascoltato i Pink Floyd ma non hanno mai ascoltato i Queen.

Una riga = un nickname presente nel primo insieme e assente nel secondo.

```
-- Ogni SELECT costruisce un insieme di nickname.
SELECT S.nickname
FROM STUDENTI S
JOIN ASCOLTI A ON A.studente = S.id_studente
JOIN BRANI B ON B.id_brano = A.brano
JOIN ARTISTI AR ON AR.id_artista = B.artista
WHERE AR.nome = 'Pink Floyd'
-- Sottrai chi ha ascoltato Queen dall'insieme Pink Floyd.
MINUS
SELECT S.nickname
FROM STUDENTI S
JOIN ASCOLTI A ON A.studente = S.id_studente
JOIN BRANI B ON B.id_brano = A.brano
JOIN ARTISTI AR ON AR.id_artista = B.artista
WHERE AR.nome = 'Queen'
-- Ordina il risultato dell'intera differenza insiemistica.
ORDER BY 1;
```

Risultato verificato: Una riga: luna.sql.

RICHIESTA 17a | IL RAGIONAMENTO

# Perché si scrive così

Leggi il percorso come una trasformazione del problema, non come una sequenza da memorizzare.

### 1. Costruisci il primo insieme

La prima SELECT individua i nickname collegati ad almeno un ascolto di un brano dei Pink Floyd.

### 2. Costruisci l'insieme da sottrarre

La seconda SELECT usa lo stesso percorso di join ma filtra gli artisti Queen.

### 3. Applica MINUS nel verso corretto

Il risultato è Pink Floyd meno Queen. Chi ha ascoltato entrambi viene tolto; chi ha ascoltato solo Queen non entra mai nel primo insieme.

### 4. Ordina il risultato dell'intera operazione

ORDER BY 1 si riferisce alla prima colonna dell'unico risultato finale, cioè il nickname.

## Scelte e alternative

MINUS invece di <> Queen sulla singola riga. Un evento Pink Floyd ha ovviamente un artista diverso dai Queen. Quel confronto non dimostra che la persona non abbia ascoltato Queen in un altro evento.

Niente DISTINCT aggiuntivo. MINUS senza ALL ha semantica insiemistica e rimuove i duplicati. Più ascolti dello stesso artista non generano più copie del nickname finale.

Alternativa EXISTS e NOT EXISTS. È equivalente chiedere che esista un ascolto Pink Floyd e non esista un ascolto Queen per lo stesso studente. MINUS rende didatticamente visibile la sottrazione di due insiemi.

> Esempio sui dati: Luna.sql ha ascoltato Wish You Were Here dei Pink Floyd e nessun brano dei Queen: è il solo risultato. Aurora ascolta Pink Floyd ma anche Queen e viene eliminata.

Errore da evitare. Invertire le due SELECT produce Queen meno Pink Floyd, una domanda diversa. I due blocchi devono restituire lo stesso numero di colonne con tipi compatibili.

Autoverifica. Uno studente che non ha ascoltato né Pink Floyd né Queen compare?

Risposta. No: manca già dal primo insieme. Non ascoltare Queen, da solo, non basta.

RICHIESTA 17b | QUERY COMMENTATA

# Sia Queen sia AC/DC

Trovare gli studenti che hanno ascoltato almeno un brano dei Queen e almeno un brano degli AC/DC.

Una riga = un nickname presente in entrambi gli insiemi di ascoltatori.

```
-- Ogni SELECT costruisce un insieme di nickname.
SELECT S.nickname
FROM STUDENTI S
JOIN ASCOLTI A ON A.studente = S.id_studente
JOIN BRANI B ON B.id_brano = A.brano
JOIN ARTISTI AR ON AR.id_artista = B.artista
WHERE AR.nome = 'Queen'
-- Conserva solo i nickname presenti in entrambi gli insiemi.
INTERSECT
SELECT S.nickname
FROM STUDENTI S
JOIN ASCOLTI A ON A.studente = S.id_studente
JOIN BRANI B ON B.id_brano = A.brano
JOIN ARTISTI AR ON AR.id_artista = B.artista
WHERE AR.nome = 'AC/DC'
-- Ordina il risultato finale, senza ripetizioni.
ORDER BY 1;
```

Risultato verificato: 5 nickname: ada.exe, bytebeat, echo, rocky, synthwave.

RICHIESTA 17b | IL RAGIONAMENTO

# Perché si scrive così

Leggi il percorso come una trasformazione del problema, non come una sequenza da memorizzare.

### 1. Seleziona gli ascoltatori dei Queen

Il primo blocco segue i join fino all'artista e restituisce i nickname degli studenti con eventi Queen.

### 2. Seleziona gli ascoltatori degli AC/DC

Il secondo blocco restituisce lo stesso tipo di dato per gli eventi dell'altro artista.

### 3. Trova l'intersezione

INTERSECT conserva soltanto i nickname presenti sia nel primo sia nel secondo risultato e rimuove le ripetizioni.

### 4. Leggi la congiunzione al livello della persona

Gli ascolti dei due artisti possono essere eventi diversi, in date diverse. La condizione e riguarda lo studente, non una singola riga di ARTISTI.

## Scelte e alternative

INTERSECT invece di AND su AR.nome. AR.nome = 'Queen' AND AR.nome = 'AC/DC' non può essere vero sulla stessa riga. Un singolo artista non può avere simultaneamente quei due nomi.

Non usare UNION per dire entrambi. UNION combina chi ha ascoltato almeno uno dei due artisti: è una condizione di alternativa, non di presenza in entrambi i gruppi.

Alternativa con due EXISTS. Due sottoquery EXISTS correlate allo stesso studente, unite da AND, esprimono la stessa condizione. Anche un raggruppamento con conteggio di due artisti distinti può funzionare se il filtro limita prima gli artisti proprio a questi due.

> Esempio sui dati: Ada.exe compare perché ha ascolti Queen e AC/DC. La presenza di più ascolti Queen non può compensare l'assenza totale di AC/DC.

Errore da evitare. GROUP BY studente HAVING COUNT(*) >= 2, da solo, è insufficiente: due ascolti dello stesso artista non dimostrano l'ascolto dell'altro.

Autoverifica. La seconda SELECT deve restituire anche il nome dell'artista?

Risposta. No: cerchiamo l'intersezione dei nickname. Aggiungere un nome artista diverso in ciascun blocco impedirebbe di trovare le stesse righe complete.

RICHIESTA 18 | QUERY COMMENTATA

# Ascolti per tutti i mesi del 2026

Mostrare il numero di ascolti per tutti i dodici mesi del 2026, includendo i mesi senza eventi con valore zero.

Una riga = uno dei dodici mesi dell'anno scelto, anche quando non ha ascolti.

```
-- Crea un calendario minimo indipendente dagli ascolti.
WITH mesi AS (
-- DUAL e CONNECT BY generano i numeri da 1 a 12.
    SELECT LEVEL AS mese FROM dual CONNECT BY LEVEL <= 12
)
-- Conta soltanto id_ascolto non nulli: i mesi vuoti danno zero.
SELECT M.mese, COUNT(A.id_ascolto) AS num_ascolti
FROM mesi M
-- Il lato sinistro contiene tutti i mesi da conservare.
LEFT JOIN ASCOLTI A
-- Abbina il mese dell'evento al numero del calendario.
    ON EXTRACT(MONTH FROM A.data_ora) = M.mese
-- Assunzione esplicita: considera soltanto l'anno 2026.
   AND A.data_ora >= TIMESTAMP '2026-01-01 00:00:00'
-- Filtro nell'ON: non cancella i mesi senza eventi.
   AND A.data_ora < TIMESTAMP '2027-01-01 00:00:00'
-- Una riga per ogni mese del calendario.
GROUP BY M.mese
-- Ordine naturale dei mesi da 1 a 12.
ORDER BY M.mese;
```

Risultato verificato: 12 righe: gennaio 0; febbraio-luglio 12 ciascuno; agosto-dicembre 0. Totale: 72 ascolti.

RICHIESTA 18 | IL RAGIONAMENTO

# Perché si scrive così

Leggi il percorso come una trasformazione del problema, non come una sequenza da memorizzare.

### 1. Genera prima il calendario minimo

La CTE mesi crea i numeri da 1 a 12 con LEVEL e CONNECT BY. Questo insieme esiste indipendentemente dai dati di ASCOLTI.

### 2. Conserva tutti i mesi

Il LEFT JOIN parte da mesi, così gennaio e gli altri mesi senza eventi non scompaiono.

### 3. Abbina mese e anno corretti

EXTRACT(MONTH FROM A.data_ora) abbina il numero del mese. I due confronti temporali nell'ON limitano gli eventi al 2026 senza eliminare le righe del calendario.

### 4. Conta soltanto gli eventi reali

COUNT(A.id_ascolto) restituisce 0 sulle righe senza abbinamento. GROUP BY M.mese produce una riga per mese e ORDER BY segue l'ordine numerico del calendario.

## Scelte e alternative

Calendario esplicito invece di solo GROUP BY su ASCOLTI. Se raggruppi soltanto gli eventi, i mesi senza eventi non esistono nel risultato: non basta aggiungere COALESCE per creare righe che non ci sono.

Filtro dell'anno nell'ON, non nel WHERE. Sulle righe conservate senza ascolti, A.data_ora è NULL. Un filtro temporale nel WHERE le eliminerebbe. Nell'ON limita soltanto gli eventi da abbinare a ciascun mese.

CONNECT BY invece della CTE ricorsiva del docente. Entrambe le forme generano correttamente dodici mesi in Oracle 21c. Il docente non filtra l'anno: qui dichiaro il 2026. Sui dati originali, tutti del 2026, i risultati coincidono; con anni ulteriori la differenza diventerebbe rilevante.

> Esempio sui dati: Gennaio non ha eventi ma il calendario contiene comunque il numero 1: il risultato è 0. Da febbraio a luglio si contano 12 ascolti per mese. Agosto-dicembre restano a zero.

Errore da evitare. COUNT(*) conta anche la riga del mese senza abbinamenti e la farebbe risultare come un ascolto. EXTRACT del solo mese mescolerebbe anni diversi se non aggiungessi il filtro sull'anno.

Autoverifica. Perché COALESCE(COUNT(...), 0) non è la soluzione principale?

Risposta. COUNT dà già zero per un gruppo senza id_ascolto non nulli. Il problema decisivo è creare e conservare tutti i mesi, anche quando non esiste alcun evento.

# Controlli, fonti e firma

## Che cosa è stato verificato

Le 19 SELECT - due nella richiesta 17 - sono state eseguite su Oracle Database 21c XE, nello schema UNINA. Il caricamento deriva dallo script Oracle originale del docente. I conteggi sono 6 artisti, 11 studenti, 18 brani e 72 ascolti.

Un programma Python separato ha ricalcolato i risultati a partire dalle righe delle quattro tabelle, confrontando valori, molteplicità e ordinamento. I risultati sono coerenti con il PDF delle soluzioni fornito dal docente. I commenti aggiunti nel SQL non cambiano le interrogazioni già verificate.

Questi controlli coprono i dati originali: non sono una dimostrazione formale per qualunque modifica futura del catalogo. I casi limite rilevanti sono dichiarati nei capitoli 6, 8, 9, 10, 11, 15 e 18.

## Riferimenti

Fonti principali fornite nello ZIP: Esercitazione SQL.pdf; Esercitazione SQL soluzioni.pdf; unina_spotify_setup_oracle.sql. Sono la base delle tracce, dei nomi e dei risultati attesi.

Repository ufficiale Oracle: schemi di esempio, customer_orders/co_create.sql

Oracle 21c SQL Language Reference: SELECT

Oracle 21c: JOIN

Oracle 21c: COUNT

Oracle 21c: EXISTS

Oracle 21c: operatori insiemistici

## Prima di considerare finito un esercizio

Sai dire che cosa rappresenta una riga del risultato? Hai verificato il percorso delle chiavi? Hai controllato duplicati, righe senza eventi, NULL e pari merito? Sai motivare una forma alternativa? Se sì, stai imparando la logica, non soltanto la sintassi.

## A cura di Antonio Ferone

antonioferone.it

Firma editoriale del materiale didattico | 22 ottobre 2026

# Appendice pratica: partire dai file del docente

Edizione ampliata del 22 ottobre 2026. I capitoli e la numerazione delle prime 43 pagine restano invariati. Questa appendice completa il percorso con ambiente, materiali originali e controlli riproducibili.

## Prima prova la traccia, poi consulta il ragionamento

Apri la traccia originale del professor Moscato e prova le richieste senza consultare subito le soluzioni. Usa le spiegazioni della guida per motivare le scelte; alla fine confronta la soluzione del docente e le alternative.

## Materiali disponibili sulla pagina della risorsa

Traccia originale, soluzioni del docente, guida Oracle/APEX e due script di preparazione del database: Oracle e SQLite. Gli originali sono mantenuti separati dalla guida ragionata di Antonio Ferone.

## Scegli il tuo ambiente

Oracle e SQL Developer seguono il percorso del corso. SQLite offre un laboratorio leggero, ma richiede adattamenti di sintassi e di alcune espressioni. Le query delle pagine precedenti sono in dialetto Oracle.

## Il controllo iniziale

```
ARTISTI    6
STUDENTI  11
BRANI     18
ASCOLTI   72
```

Pagina web e download: antonioferone.it/risorse/esercitazione-sql-unina-spotify/

# Preparare il laboratorio Oracle

## 1. Collega SQL Developer al database

Nel laboratorio Oracle XE 21c locale: host localhost, porta 1521, Nome servizio XEPDB1. Sono i valori di questo ambiente didattico: verifica quelli della tua installazione. La connessione amministrativa serve soltanto per creare l'utente del laboratorio.

## 2. Crea un utente dedicato

Collegati al servizio XEPDB1 con un amministratore del tuo database locale. Sostituisci il segnaposto della password prima di eseguire il comando. Se UNINA esiste già, controllane il contenuto: non ricrearlo o cancellarlo automaticamente.

```
CREATE USER unina
  IDENTIFIED BY "INSERISCI_UNA_PASSWORD_PERSONALE"
  QUOTA 100M ON USERS;
GRANT CREATE SESSION, CREATE TABLE,
  CREATE VIEW, CREATE SEQUENCE TO unina;
```

## 3. Importa i dati con UNINA

Crea una nuova connessione per UNINA, apri unina_spotify_setup_oracle.sql e usa Esegui script (F5). Lo script crea le tabelle, i vincoli, la sequenza e i dati. Va usato in uno schema di laboratorio vuoto; non include un ripristino automatico.

## 4. Controlla i conteggi

```
SELECT 'ARTISTI' AS tabella, COUNT(*) AS righe FROM ARTISTI
UNION ALL SELECT 'STUDENTI', COUNT(*) FROM STUDENTI
UNION ALL SELECT 'BRANI', COUNT(*) FROM BRANI
UNION ALL SELECT 'ASCOLTI', COUNT(*) FROM ASCOLTI;
```

I risultati attesi sono 6, 11, 18 e 72. Per le SELECT usa Ctrl+Invio sull'istruzione corrente. APEX, ORDS e Tomcat appartengono al percorso web avanzato, distinto da questo laboratorio SQL.

# Un percorso di difficoltà crescente

La traccia del docente è composta da 18 richieste. Nella guida la richiesta 17 è divisa in due parti, quindi i blocchi SQL sono 19.

## Livello 1 - JOIN e filtri: richieste 1-3

Collega le chiavi corrette, combina i filtri e conserva il dettaglio dei singoli ascolti. Nella richiesta 2 le righe ripetute possono rappresentare eventi distinti.

## Livello 2 - Aggregazioni: richieste 4-6

Distingui la granularità del risultato: un genere, un artista, un brano. WHERE filtra righe; HAVING filtra gruppi. Nella richiesta 6 il rapporto deve essere calcolato prima della media.

## Livello 3 - JOIN esterni: richieste 7-8

Mantieni anche chi non ha eventi. Un conteggio della chiave dell'evento produce zero sulle righe senza abbinamento; COUNT(*) non ha lo stesso significato.

## Livello 4 - Massimi e pari merito: richieste 9-11

Trovare il massimo non significa scegliere una riga arbitraria. La richiesta 11 contiene due studenti a pari merito.

## Livello 5 - Sottoquery: richieste 12-15

Almeno uno, nessuno e tutti sono condizioni diverse. Collega sempre le sottoquery alla persona o al brano giusto.

## Livello 6 - Sfide: richieste 16-18

Crea una vista di dettaglio, ragiona per insiemi e genera un calendario che conservi anche i mesi senza ascolti.

# Confrontare le soluzioni

## Richiesta 1: virgola e JOIN esplicito

Il docente collega BRANI e ARTISTI con la relazione nel WHERE. La guida usa JOIN ... ON. Le forme esprimono lo stesso collegamento in questo caso: la seconda rende visibile la distinzione tra relazione e filtro.

## Richiesta 3: EXTRACT e intervallo temporale

Il docente filtra anno e mese con EXTRACT; la guida usa un inizio incluso e una fine esclusa. Entrambe descrivono aprile 2026 sui dati forniti. Una valutazione delle prestazioni richiederebbe indici e piano di esecuzione.

## Richiesta 12: IN ed EXISTS

Il docente usa IN sull'insieme degli studenti che hanno ascoltato artisti italiani. La guida usa EXISTS correlato. In questa richiesta entrambe cercano almeno un evento qualificante, senza ripetere lo studente per ogni ascolto.

## Richiesta 18: due calendari, un'assunzione da esplicitare

Il docente genera i mesi con una CTE ricorsiva; la guida usa LEVEL e CONNECT BY. La guida limita anche gli ascolti al 2026, mentre il filtro del docente considera il numero del mese senza un anno esplicito. Sul dataset, che contiene ascolti del 2026, il conteggio coincide. Su dati di più anni la domanda va precisata.

## Il criterio per confrontare

Confronta valori, duplicati, ordinamento richiesto e casi limite. Una sintassi differente non prova che una soluzione sia errata; risultati uguali sul dataset non dimostrano equivalenza su tutti i dati possibili.

# Provare l’esercitazione con SQLite

## Un file separato per il database

Usa unina_spotify_setup_sqlite.sql in un database SQLite nuovo. I dati rappresentano lo stesso caso didattico, ma i tipi, gli identificatori automatici e le funzioni non sono quelli di Oracle.

## Attenzione alla divisione della richiesta 6

```
AVG(1.0 * A.secondi_ascoltati / B.durata_secondi)
```

Il fattore 1.0 fa usare un rapporto reale. In SQLite la divisione tra due interi può troncare la parte frazionaria: aggiungere AVG all’esterno non recupera la precisione già persa.

## Le altre differenze

MINUS diventa EXCEPT. I filtri temporali usano le date del dataset nel formato ISO. L’estrazione del mese si può esprimere con strftime. Il calendario usa una CTE ricorsiva invece di CONNECT BY; la creazione della vista usa CREATE VIEW.

## Le query adattate sono disponibili a parte

Il file query_sqlite_commentate_af.sql contiene i 19 blocchi adattati e verificati sui dati forniti. È un adattamento di Antonio Ferone, distinto dagli script originali del docente.

## Che cosa è stato controllato in questa integrazione

Lo script SQLite è stato eseguito in un database locale in memoria. Sono stati verificati i conteggi, l’assenza di violazioni delle chiavi esterne e l’esecuzione dei 19 blocchi adattati. Oracle e lo stack APEX non sono stati reinstallati o rieseguiti per questa integrazione.

# Fonti, materiali e percorso successivo

## Attribuzione dei materiali

Traccia e soluzioni: professor Moscato, come indicato da Antonio Ferone. Guida ragionata, confronti e adattamento SQLite delle query: Antonio Ferone. La guida Oracle/APEX è distribuita come documento di accompagnamento ricevuto con il materiale del corso.

## Oracle/APEX: una risorsa collegata

La guida di accompagnamento descrive un ambiente Windows con Oracle XE 21c, SQL Developer e uno stack APEX 22.2, ORDS 22.4.4, JDK 17 e Tomcat 9.0.73. È un riferimento per quello specifico laboratorio, non una verifica di compatibilità di tutte le release attuali.

Per una nuova installazione confronta i requisiti delle versioni scelte nelle documentazioni ufficiali. Le password nei documenti sono esempi di laboratorio: scegli credenziali personali, senza riutilizzare gli esempi per servizi pubblici.

## Documentazione ufficiale

Oracle Database XE 21c - Installazione Windows

Oracle APEX 22.2 - Requisiti dell'ambiente del documento

Oracle APEX 26.1 - Requisiti della release consultata

SQLite - Espressioni e aritmetica

## Prima di passare oltre

Sai caricare i dati, spiegare cosa rappresenta una riga e motivare il risultato? Se sì, puoi affrontare gli esercizi cambiando i dati e controllando se il tuo ragionamento continua a funzionare.