Fondamenti dei database relazionali e SQL

Obiettivi di apprendimento

Al termine di questa unità, gli studenti saranno in grado di:

- Spiegare i concetti fondamentali dei database relazionali e il loro ruolo nella gestione dei dati strutturati.
- Descrivere i componenti principali dei sistemi di database: tabelle, relazioni, chiavi e vincoli.
- Comprendere i principi della normalizzazione e applicarli per migliorare la progettazione dei database.
- Scrivere interrogazioni SQL usando SELECT, WHERE, GROUP BY, HAVING, JOIN, ORDER BY, INSERT, UPDATE e DELETE.
- Progettare schemi relazionali di base per dati ambientali (ad esempio, qualità dell'aria e monitoraggio del rumore).
- Analizzare dati ambientali con SQL (aggregazioni, filtri, join e sottoquery).
- Interpretare i risultati delle interrogazioni e preparare brevi relazioni analitiche.

Contenuti della lezione

Introduzione

Nei sistemi digitali i dati costituiscono una risorsa fondamentale. Agenzie ambientali, gruppi di ricerca e istituzioni pubbliche raccolgono grandi quantità di dati osservativi: misurazioni atmosferiche, rilevazioni meteorologiche, indicatori della qualità dell'acqua, inventari delle emissioni e altro ancora. Senza un approccio strutturato, questi dati diventano frammentati, incoerenti e difficili da analizzare.

I database relazionali organizzano le informazioni in tabelle strutturate, ciascuna delle quali rappresenta un determinato tipo di entità (ad esempio, Stazioni, Misurazioni, Utenti).

Il modello dei dati si basa sul modello relazionale introdotto da E. F. Codd, che garantisce coerenza e rigore matematico.

Caratteristiche principali:

1. Struttura basata su tabelle

- I dati vengono memorizzati in tabelle (relazioni), composte da righe (tuple) e colonne (attributi).
- Ogni tabella ha uno schema definito, che specifica i nomi degli attributi, i tipi di dato e i vincoli.

2. Schema rigoroso

- Tutti i dati devono rispettare la struttura prestabilita.
- Ogni colonna ha un dominio definito (numero intero, testo, data, ecc.).
- Vincoli quali PRIMARY KEY, FOREIGN KEY, NOT NULL, UNIQUE e CHECK garantiscono l'integrità dei dati.

3. Dati strutturati e coerenti

- I record devono rispettare esattamente lo schema. Ad esempio, una tabella AirQuality(station_id, datetime, pollutant, value) richiede che ogni riga contenga questi attributi con i tipi di dato corretti.

4. Relazioni tra tabelle

- Le tabelle possono fare riferimento l'una all'altra mediante chiavi, consentendo una progettazione normalizzata ed evitando ridondanze.

Grazie a questa struttura, i database relazionali sono particolarmente adatti a dati strutturati e ben definiti, per i quali la coerenza è essenziale, come sistemi finanziari, archivi scientifici, gestione delle scorte e monitoraggio ambientale.

Panoramica delle tecnologie

Concetti fondamentali dei database relazionali

Tabelle e schemi

Nel modello relazionale tutti i dati sono organizzati in strutture logiche chiamate tabelle, formalmente dette relazioni (Tabella 1). Una tabella rappresenta un unico tipo di entità, come una stazione di monitoraggio, una misurazione dell'inquinamento o un'osservazione meteorologica, e offre un modo uniforme e strutturato per memorizzarne le diverse istanze. Ogni tabella è composta da righe e colonne. Le righe, o tuple, rappresentano singoli record; le colonne, o attributi, descrivono le proprietà di ciascun record.

Tabella 1. Elementi del modello relazionale e loro rappresentazione

Elemento del modello relazionale	Forma di rappresentazione
Relazione	Tabella
Tupla	Riga di una tabella
Attributo	Intestazione di una colonna
Chiave	Insieme di attributi che identifica univocamente ogni riga, viene usato per collegare tabelle oppure migliora l'efficienza delle operazioni
Dominio	Insieme di tutti i possibili valori di un attributo
Schema di relazione	Elenco dei nomi degli attributi che definiscono la struttura della relazione

Si consideri, ad esempio, una tabella di un sistema di monitoraggio ambientale:

AirQuality(station_id, datetime, pollutant, value, unit).

Ogni riga di AirQuality rappresenta una singola misurazione effettuata in un determinato momento, presso una determinata stazione e per uno specifico inquinante. La tabella impone una struttura coerente: tutte le misurazioni devono includere l'identificativo della stazione, una marca temporale, il nome dell'inquinante, un valore numerico e la relativa unità di misura.

La struttura di una tabella è definita da uno **schema**. Lo schema specifica gli attributi della tabella, i relativi tipi di dato e le regole che devono rispettare. Costituisce il progetto della base di dati e assicura che le informazioni memorizzate siano significative e interpretabili. In senso più ampio, lo schema può descrivere l'intero database, indicando tutte le tabelle e le relazioni e i vincoli che le collegano. Questa struttura esplicita è uno dei principali punti di forza dei database relazionali: garantisce prevedibilità, riduce le ambiguità e supporta interrogazioni e analisi affidabili.

Ad esempio, lo schema di un database per il monitoraggio ambientale può contenere:

- *Stations (station_id, name, region, latitude, longitude)*
- *AirQuality (station_id, datetime, pollutant, value, unit)*
- *Weather (station_id, datetime, temperature, humidity)*

Chiavi

Il modello relazionale si basa in larga misura sul concetto di chiave per identificare univocamente i record e mantenere collegamenti logici tra le tabelle.

Una chiave primaria (Primary Key, PK) è un attributo, o una combinazione di attributi, che identifica in modo univoco ogni riga di una tabella. Due righe non possono avere lo stesso valore di chiave primaria e una chiave primaria non può contenere NULL. Per esempio, la tabella Stations può usare station_id come chiave primaria, garantendo che ogni stazione abbia un identificativo univoco. In tabelle più complesse, come AirQuality, l'univocità può essere garantita da una chiave composta, ad esempio (station_id, datetime, pollutant).

Una chiave esterna (Foreign Key, FK) crea una relazione logica tra due tabelle. È un attributo di una tabella che fa riferimento alla chiave primaria di un'altra tabella, garantendo la coerenza dei dati. Per esempio, ogni station_id in AirQuality deve corrispondere a uno station_id esistente in Stations. Senza chiavi esterne, il database potrebbe contenere riferimenti non validi, come misurazioni relative a stazioni inesistenti o eliminate.

Le chiavi candidate sono attributi che potrebbero essere scelti come chiave primaria, poiché permettono di identificare univocamente le righe. Una tabella può avere più chiavi candidate, ma soltanto una viene scelta come chiave primaria. Le chiavi composte, invece, sono costituite da più attributi e vengono spesso utilizzate quando un solo attributo non garantisce l'univocità.

Le chiavi sono essenziali per l'efficienza, l'affidabilità e la corretta interpretazione dei dati. Supportano l'indicizzazione, impediscono le duplicazioni e consentono di collegare le tabelle: funzionalità indispensabili nei grandi database ambientali.

Relazioni

Le relazioni descrivono come i dati di una tabella siano collegati ai dati di un'altra. Il modello relazionale prevede diversi tipi di relazioni binarie, corrispondenti a differenti associazioni del mondo reale.

- **Uno a uno (1:1)**
  In una relazione uno a uno, ogni record di una tabella è collegato al massimo a un record di un'altra tabella. Questa struttura è relativamente rara, ma utile quando si vogliono separare informazioni facoltative, riservate o consultate di rado.

- **Uno a molti (1:N)**

  La relazione uno a molti è la più comune e pratica nei database relazionali. In questa struttura, una singola riga di una tabella può essere associata a più righe di un'altra.

  Ad esempio, una stazione di monitoraggio ambientale può raccogliere centinaia di misurazioni della qualità dell'aria ogni giorno. In questo caso:

  Stations si trova sul lato «uno», mentre AirQualityMeasurments si trova sul lato «molti».

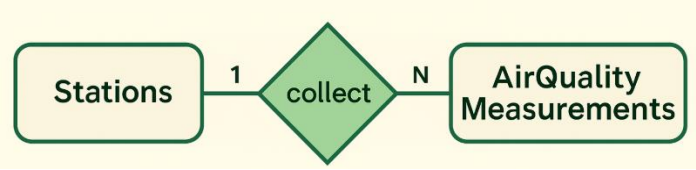


Fig. 1. Esempio di diagramma entità-relazione in notazione di Chen per una relazione 1:N

  Queste relazioni consentono di evitare ridondanze: i metadati della stazione vengono memorizzati una sola volta, ma possono essere richiamati più volte.

- **Molti a molti (M:N)**

  In una relazione molti a molti, più righe di una tabella possono essere associate a più righe di un'altra. Nei database relazionali tale relazione non viene implementata direttamente, ma mediante una tabella di associazione che memorizza le coppie di chiavi esterne.

  Ad esempio, se una stazione può monitorare più inquinanti e un inquinante può essere misurato in più stazioni, la relazione è molti a molti (Fig. 2). Una tabella di associazione come StationPollutant(station_id, pollutant) rappresenta questi collegamenti.

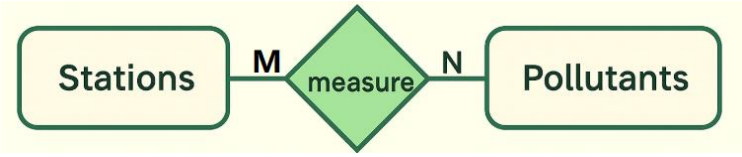


Fig. 2. Esempio di diagramma entità-relazione in notazione di Chen per una relazione M:N

Comprendere le relazioni è fondamentale per progettare schemi normalizzati e scrivere interrogazioni SQL corrette, soprattutto quelle che utilizzano i join.

La cardinalità di una relazione indica il numero di istanze di un'entità che possono essere associate a una singola istanza di un'altra entità. Nella notazione di Chen, la cardinalità è rappresentata mediante numeri posti accanto alle entità. Indica quindi quante istanze dell'entità collegata possono partecipare alla relazione.

Attributi

Un attributo rappresenta una proprietà o caratteristica di un'entità. Descrive un aspetto dell'entità e assume valori appartenenti a un insieme predefinito. L'insieme dei possibili valori di un attributo è chiamato dominio.

Nella progettazione concettuale dei database si distinguono diversi tipi di attributi in base alla struttura, al significato e al comportamento. I principali tipi sono illustrati nella Fig. 3 e descritti nella Tabella 2.

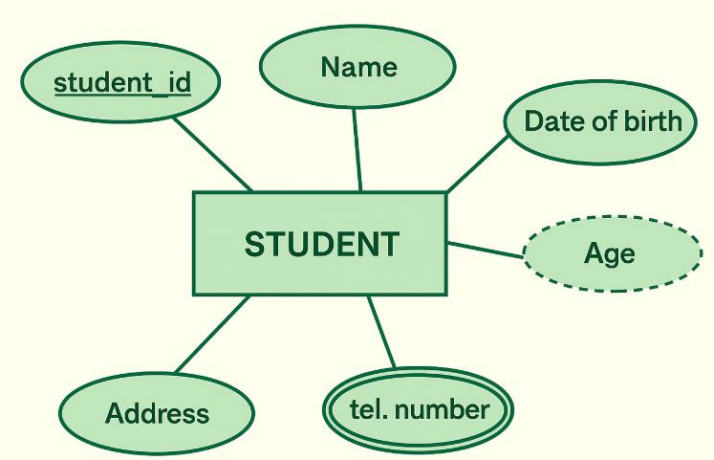


Fig. 3. Esempio di diagramma entità-relazione in notazione di Chen per l'entità Studente

Tabella 2. Tipi di attributi

Tipo di attributo	Descrizione	Esempi
Attributo semplice (atomico)	Attributo che non può essere suddiviso in componenti significative più piccole.	*student_id, cognome*

Attributo composto	Attributo suddivisibile in sottoattributi, ognuno dotato di significato autonomo.	*Indirizzo → via, numero civico, appartamento; nome completo → cognome, nome*
Attributo monovalore	Attributo che assume un solo valore per ciascuna istanza dell'entità.	*student_id, numero di passaporto*
Attributo multivalore	Attributo che può assumere più valori per la stessa istanza dell'entità.	*Numeri di telefono (cellulare, casa), indirizzi email*
Attributo derivato	Attributo il cui valore viene calcolato a partire da altri dati e non è memorizzato direttamente.	*Età (dalla data di nascita), numero di crediti (dall'elenco dei corsi)*
Attributo chiave	Attributo o insieme di attributi che identifica univocamente un'istanza dell'entità.	*Chiave primaria: matricola; chiavi candidate: email, numero del libretto*

Vincoli di integrità

I vincoli di integrità sono regole applicate automaticamente dal database per preservare accuratezza, coerenza e affidabilità. Impediscono l'inserimento di dati errati e garantiscono la validità delle relazioni tra tabelle. Tra i vincoli più comuni rientrano:

- **NOT NULL**
  Il vincolo NOT NULL richiede che una colonna contenga sempre un valore. È essenziale per gli attributi necessari all'analisi o all'identificazione. Per esempio, le marche temporali delle misurazioni ambientali non devono essere NULL.
- **UNIQUE**
  Il vincolo UNIQUE assicura che due righe non abbiano lo stesso valore per un determinato attributo. È utile per identificativi come numeri di serie, nomi utente o codici degli inquinanti.
- **CHECK**
  I vincoli CHECK assicurano che i valori degli attributi soddisfino una condizione logica.

Ad esempio, CHECK (value ≥ 0) impedisce di registrare concentrazioni negative di inquinanti, che sarebbero scientificamente non valide.

- **Integrità referenziale (tramite chiavi esterne)**
  Questo vincolo garantisce che ogni valore di chiave esterna corrisponda a un valore di chiave primaria esistente nella tabella referenziata. Impedisce inoltre l'eliminazione di righe referenziate, a meno che i record dipendenti non siano gestiti adeguatamente. L'integrità referenziale è fondamentale per mantenere coerenti i dati complessi quando le tabelle sono interconnesse da chiavi e relazioni.

Fasi della progettazione di database

La progettazione di un database è un processo strutturato in più fasi, che trasforma le esigenze informative del mondo reale in un sistema efficiente e affidabile di gestione dei dati. Pur con alcune differenze tra metodologie, l'approccio classico prevede quattro fasi fondamentali:

1. Analisi del sistema e del dominio applicativo
2. Progettazione concettuale
3. Progettazione logica
4. Progettazione fisica

Ogni fase riguarda un diverso livello di astrazione e contribuisce alla creazione di un database solido, scalabile e ben strutturato.

La progettazione inizia con l'analisi del dominio applicativo, che costituisce la base di tutte le fasi successive. In questa fase il progettista studia il contesto reale in cui opererà il database. Occorre esaminare nel dettaglio i processi organizzativi, il dominio scientifico o il contesto operativo per determinare quali informazioni il sistema debba conservare e come saranno utilizzate. L'analisi comprende l'individuazione dei portatori di interesse, ossia persone, gruppi o uffici che utilizzeranno il database, e la definizione di obiettivi, esigenze informative e aspettative. A tal fine si esaminano documenti, relazioni, moduli, flussi di lavoro e strumenti tecnologici esistenti, analizzando come i dati vengono prodotti, elaborati, trasmessi e utilizzati. Il risultato è una descrizione completa e strutturata del dominio: oggetti principali, proprietà, relazioni e regole che ne determinano il comportamento. Questa comprensione costituisce il fondamento concettuale dell'intero processo di progettazione.

Dopo aver analizzato il dominio, si passa alla progettazione concettuale, che rappresenta la struttura dei dati a un elevato livello di astrazione, indipendentemente dalla tecnologia di database.

L'obiettivo è costruire un modello concettuale, generalmente espresso tramite diagrammi entità-relazione (ER) o diagrammi delle classi UML, che rifletta correttamente la semantica del dominio. Il progettista individua le entità, ne definisce gli attributi e stabilisce le relazioni tra esse, comprese cardinalità e partecipazione facoltativa. La progettazione concettuale incorpora anche le regole individuate durante l'analisi, affinché il futuro database rispecchi vincoli e processi reali. La domanda centrale è quali informazioni debbano essere rappresentate, senza ancora considerare come saranno implementate o memorizzate.

La fase successiva, la progettazione logica, trasforma il modello concettuale in una struttura adatta a un determinato tipo di sistema, di norma un DBMS relazionale. Dipende dal modello dei dati scelto, ma non dalle modalità fisiche di memorizzazione. Il modello concettuale viene tradotto in tabelle, chiavi primarie, chiavi candidate, chiavi esterne e attributi con i rispettivi tipi di dato. La progettazione logica comprende anche la normalizzazione, che elimina sistematicamente le ridondanze e previene anomalie di aggiornamento, inserimento e cancellazione. Mediante vincoli e regole di integrità, lo schema logico garantisce l'accuratezza e la coerenza dei dati. Questa fase risponde quindi alla domanda di come realizzare la rappresentazione concettuale nel modello dei dati del DBMS scelto.

La fase finale è la progettazione fisica, relativa all'implementazione concreta del modello logico in uno specifico ambiente hardware e software. Si concentra su prestazioni, efficienza e strategie di gestione. Il progettista definisce come i dati saranno memorizzati e consultati, scegliendo strutture di archiviazione, organizzazione dei file, indici, partizionamento e clustering in funzione dei carichi di lavoro previsti. Vengono considerate anche l'allocazione della memoria, l'ottimizzazione delle interrogazioni e dei percorsi di accesso, nonché le procedure di backup, archiviazione e ripristino. Possono essere introdotte misure di sicurezza quali controllo degli accessi, cifratura e audit. La progettazione fisica adatta dunque lo schema logico ai vincoli operativi e alle caratteristiche prestazionali del DBMS, rispondendo alla domanda di come memorizzare e accedere ai dati in modo efficiente.

Tabella 3. Sintesi delle fasi di progettazione di un database

Fase	Domanda fondamentale	Risultato
Analisi del sistema	*Qual è il dominio? Di cosa hanno bisogno gli utenti?*	Requisiti, regole operative, comprensione del dominio
Progettazione concettuale	*Un attributo che può essere suddiviso in sottoattributi con significato autonomo.*	Diagramma ER o schema concettuale
Progettazione logica	*Come rappresentare i dati nel modello di un DBMS?*	Schema relazionale: tabelle, chiavi e vincoli
Progettazione fisica	*Come memorizzare e accedere ai dati in modo efficiente?*	Indici, partizioni, configurazione dell'archiviazione

Normalizzazione

La normalizzazione è un metodo sistematico di progettazione dei database relazionali che organizza i dati in tabelle ben strutturate per ridurre le ridondanze e prevenire incoerenze. Mira a conservare ciascun fatto una sola volta e a far dipendere ogni attributo dalla chiave, dall'intera chiave e da nient'altro che la chiave. Migliora la chiarezza, riduce lo spreco di spazio e protegge dalle anomalie di aggiornamento, inserimento e cancellazione che possono alterare le analisi.

Sebbene la normalizzazione sia un concetto teorico radicato nella teoria degli insiemi e nell'algebra relazionale, la sua utilità è particolarmente evidente nella gestione dei dati ambientali. Le misurazioni della qualità dell'aria, le osservazioni meteorologiche, le categorie di inquinanti e i metadati delle stazioni sono spesso voluminosi, ripetitivi e interdipendenti. Senza normalizzazione, la duplicazione delle informazioni sulle stazioni, delle definizioni degli inquinanti e delle unità di misura può generare rapidamente incoerenze nei dati e nei processi di rendicontazione.

Per eliminare sistematicamente le ridondanze, la normalizzazione procede attraverso una successione di forme normali, ciascuna caratterizzata da requisiti più rigorosi. Le più importanti nella pratica sono la prima, la seconda e la terza forma normale.

Prima forma normale (1NF)

La prima forma normale richiede che tutti i valori di una tabella siano atomici, cioè che ogni cella contenga un solo valore indivisibile. Non sono ammessi gruppi ripetuti, tabelle annidate o più valori in uno stesso campo.

Ad esempio, un archivio ambientale progettato male potrebbe contenere più inquinanti in una sola colonna:

station_id	pollutants_measured
101	NO2, PM2.5, O3

Una rappresentazione del genere rende molto difficili le interrogazioni, i filtri e l'indicizzazione. In 1NF, ogni inquinante deve essere riportato in una riga distinta oppure trasferito in una tabella collegata. Dopo la normalizzazione, la struttura risulta chiara, interrogabile e coerente:

station_id	pollutants_measured
101	NO2
101	PM2.5
101	O3

Garantendo l'atomicità, la 1NF pone le basi per tutte le forme normali successive.

Seconda forma normale (2NF)

La seconda forma normale si applica alle tabelle con chiavi primarie composte, formate da più attributi. Richiede che ogni attributo non appartenente alla chiave dipenda dall'intera chiave primaria, e non soltanto da una sua parte. Non sono quindi ammesse dipendenze parziali.

Si consideri una tabella di misurazioni della qualità dell'aria identificata dalla chiave (station_id, datetime):

Si applica alle tabelle con chiavi primarie composte, formate da più attributi. Richiede che ogni attributo non appartenente alla chiave dipenda dall'intera chiave primaria, e non soltanto da una sua parte. Non sono ammesse dipendenze parziali.

Si consideri una tabella di misurazioni della qualità dell'aria identificata dalla chiave (station_id, datetime):

station_id	**datetime**	**pollutant**	**value**	**station_name**

101	2025-01-15 09:00	NO2	40	Zagreb Center

La 2NF richiede di spostare questi attributi in una tabella separata, in questo caso Stations. Dopo la normalizzazione:

AirQualityMeasurements(station_id, datetime, pollutant, value)

Stations(station_id, station_name)

Questa struttura elimina le ridondanze e garantisce che ogni attributo sia collocato nella tabella appropriata.

Terza forma normale (3NF)

La terza forma normale elimina le dipendenze transitive: nessun attributo non chiave deve dipendere da un altro attributo non chiave. Gli attributi devono dipendere soltanto dalla chiave primaria.

Per illustrare la terza forma normale con dati ambientali, si consideri una tabella contenente informazioni sulle stazioni meteorologiche e sulle regioni a cui appartengono:

station_id	**station_name**	**region_name**	**region_manager**
201	Riverbank West	Central Basin	Dr. Alicia Moreno
202	Hilltop South	Mountain Highlands	Eng. Robert Davies

La chiave primaria è station_id. Gli attributi station_name, region_name e region_manager non fanno parte della chiave. region_manager non dipende direttamente da station_id, ma da region_name, un altro attributo non chiave. Si crea così una dipendenza transitiva:

station_id → region_name

region_name → region_manager.

Questo viola la terza forma normale.

Per eliminare la dipendenza transitiva e raggiungere la terza forma normale, suddividiamo la tabella in due tabelle collegate:

station_id (PK)	**station_name**	**region_name**

201	Riverbank West	Central Basin
202	Hilltop South	Mountain Highlands

region_name (PK)	**region_manager**
Central Basin	Dr. Alicia Moreno
Mountain Highlands	Eng. Robert Davies

Il database è ora in terza forma normale:

- *station_id identifica univocamente ogni stazione.*
- *region_name identifica univocamente ogni regione.*
- region_manager dipende soltanto dalla propria chiave (region_name).
- Nessun attributo non chiave dipende da un altro attributo non chiave.

La normalizzazione è fondamentale per preservare integrità e affidabilità, evitando le anomalie logiche che si verificano quando i dati sono conservati in modo non strutturato o ridondante. Queste anomalie nascono quando uno stesso fatto compare ripetutamente in più righe o quando informazioni di diversa natura vengono erroneamente riunite in un'unica tabella. Nei sistemi informativi ambientali, dove misurazioni, metadati delle stazioni, classificazioni regionali e dati normativi interagiscono continuamente, tali incoerenze possono compromettere rapidamente l'accuratezza delle analisi e dei rapporti.

Un problema comune è l'anomalia di aggiornamento, che si verifica quando una modifica deve essere applicata a informazioni ripetute. Se il nome di una stazione o di una regione compare in molte righe, tutte le occorrenze devono essere aggiornate in modo coerente. Dimenticarne una può produrre record contraddittori: la stessa stazione può apparire con nomi diversi o una regione con responsabili differenti. Tali incoerenze si propagano nei report, nelle dashboard e nei modelli successivi.

L'anomalia di inserimento si verifica quando non è possibile aggiungere un'informazione perché mancano altre informazioni indipendenti. Ad esempio, potrebbe risultare impossibile registrare un nuovo inquinante o una nuova regione ambientale finché non esista almeno una misurazione

associata. Ciò impone un ordine artificiale nella creazione dei dati e ostacola i normali flussi di lavoro.

L'anomalia di cancellazione è il problema opposto: eliminare l'ultima occorrenza di un dato operativo può cancellare involontariamente importanti informazioni descrittive. Se i metadati della stazione sono conservati insieme alle misurazioni, l'eliminazione dell'ultima misurazione potrebbe far scomparire anche la descrizione della stazione, sebbene questa esista ancora nella realtà.

Organizzando sistematicamente le informazioni in tabelle normalizzate, queste anomalie vengono eliminate. Ogni fatto è conservato una sola volta, nella tabella logicamente corretta, mentre le altre tabelle lo richiamano tramite chiavi anziché duplicarlo. Il database diventa così più coerente, più facile da mantenere e meno vulnerabile ad aggiornamenti o cancellazioni errati. Nelle applicazioni ambientali, come il monitoraggio a lungo termine dell'aria, la valutazione dei bacini idrografici e la modellazione ecologica, la normalizzazione mantiene i dati affidabili, tracciabili e adatti alle analisi. Tutela la continuità delle osservazioni scientifiche, favorisce il rispetto delle norme e fornisce una base stabile per la modellazione ambientale e le decisioni su larga scala.

Linguaggio SQL (Structured Query Language)

SQL è il linguaggio standard per interagire con i sistemi di gestione di database relazionali. Permette di definire la struttura dei dati, modificare le informazioni memorizzate, regolare gli accessi agli oggetti del database e gestire le transazioni. Sebbene sia noto soprattutto per l'estrazione dei dati, SQL è un linguaggio articolato in diversi sottolinguaggi specializzati, ciascuno con una funzione specifica nelle operazioni sul database. Tradizionalmente si distinguono quattro categorie principali: DDL, DML, DCL e TCL.

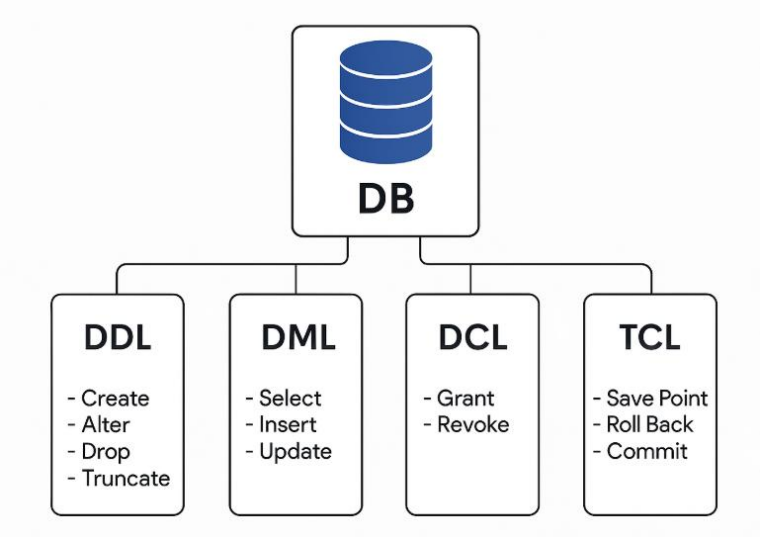


Fig. 4. Struttura di SQL

1. Linguaggio di definizione dei dati (DDL)

Il DDL descrive e modifica la struttura degli oggetti del database. Le istruzioni DDL definiscono lo schema, cioè l'architettura del database, specificando quali tabelle esistono, quali attributi contengono e come sono collegate.

- CREATE consente di creare un nuovo oggetto del database, come una tabella, una vista, un indice o persino un intero database. Si utilizza, ad esempio, per definire una nuova tabella di misurazioni ambientali o una vista che riepiloga l'andamento degli inquinanti.

```
CREATE TABLE Stations (
    station_id INT PRIMARY KEY,
    name VARCHAR(100),
    region VARCHAR(50)
);
```

- **ALTER modifica un oggetto esistente del database. Può aggiungere una colonna quando sono necessari ulteriori attributi ambientali, modificare un tipo di dato per accogliere misurazioni più precise oppure cambiare i vincoli per applicare nuove regole.**

```
ALTER TABLE AirQuality ADD COLUMN unit VARCHAR(20);
```

- DROP elimina completamente un oggetto del database. Quando una tabella, un indice o una vista non sono più necessari, ad esempio dopo una riprogettazione o il consolidamento dei dati, DROP li rimuove definitivamente dallo schema.

```
DROP TABLE TempData;
```

Le istruzioni DDL definiscono la struttura fondamentale del database, sulla quale si basano la memorizzazione e l'accesso ai dati.

2. Linguaggio di manipolazione dei dati (DML)

Mentre il DDL costruisce la struttura, il DML gestisce le operazioni quotidiane sui dati. Questi comandi permettono di inserire nuove informazioni, recuperare record esistenti, modificarli e rimuovere dati obsoleti o errati.

- **INSERT aggiunge nuovi record a una tabella. Quando arrivano nuove letture dei sensori o vengono registrati metadati di ulteriori stazioni, le istruzioni INSERT popolano il database.**

```
INSERT INTO Stations VALUES (101, 'Zagreb-Center', 'Zagreb');
```

- UPDATE modifica i record esistenti. Se cambiano i metadati di una stazione, ad esempio l'altitudine, le coordinate o l'operatore, UPDATE permette di mantenere le informazioni corrette.

```
UPDATE AirQuality
SET value = NULL
WHERE value < 0;
```

- **DELETE rimuove i record da una tabella. Viene utilizzato per eliminare inserimenti errati o dati non più necessari.**

```
DELETE FROM Stations WHERE region = 'Unknown';
```

- **SELECT recupera dati da una o più tabelle, eventualmente applicando filtri, ordinamenti, raggruppamenti e aggregazioni. Gli analisti ambientali possono usarlo per estrarre misurazioni che superano i limiti normativi o confrontare livelli di inquinamento tra stazioni.**

```
SELECT station_id, pollutant, value
FROM AirQuality
WHERE pollutant = 'PM2.5';
```

  - ***Ordinamento***

```
ORDER BY value DESC;
```

  - ***Filtraggio con operatori logici***

```
WHERE value > 50 AND pollutant = 'NO2';
```

  - ***Aggregazione e GROUP BY (utilizzati nei rapporti ambientali)***

```
SELECT station_id, pollutant, AVG(value) AS avg_value
FROM AirQuality
GROUP BY station_id, pollutant;
```

  - ***HAVING: filtraggio dei dati aggregati***

```
HAVING AVG(value) > 40;
```

  - ***INNER JOIN***

```
SELECT s.name, a.pollutant, a.value
FROM AirQuality a
JOIN Stations s ON a.station_id = s.station_id;
```

  - ***LEFT JOIN***

    Scenario: vogliamo elencare tutte le stazioni di monitoraggio, comprese quelle che non hanno ancora misurazioni registrate. Tabella 1: Stations(station_id, station_name). Tabella 2: AirQualityMeasurements(station_id, pollutant, value).

```
SELECT s.station_id, s.station_name, m.pollutant, m.value
FROM Stations AS s
LEFT JOIN AirQualityMeasurements AS m
ON s.station_id = m.station_id;
```

  - ***RIGHT JOIN***

    Scenario: vogliamo elencare tutti gli inquinanti definiti, anche se alcuni non sono ancora associati a nessuna stazione. Tabella 1:

AirQualityMeasurements(station_id, pollutant, value). Tabella 2: Pollutants(pollutant, description).

```
SELECT m.station_id, p.pollutant, p.description, m.value
FROM AirQualityMeasurements AS m
RIGHT JOIN Pollutants AS p
ON m.pollutant = p.pollutant;
```

- **Sottoquery**

  Analisi iterative e annidate, ad esempio per individuare superamenti dei limiti dell'UE:

```
SELECT *
FROM AirQuality
WHERE value > (
      SELECT threshold
      FROM EU_Standards
      WHERE pollutant = AirQuality.pollutant
);
```

Il DML costituisce il nucleo operativo di SQL e permette di lavorare direttamente sulle informazioni memorizzate.

3. Linguaggio di controllo dei dati (DCL)

Il DCL regola l'accesso agli oggetti del database e gestisce i permessi. Negli ambienti multiutente, specialmente nei sistemi scientifici e pubblici in cui i dati ambientali devono essere condivisi ma anche protetti, il DCL assicura che soltanto le persone autorizzate possano svolgere determinate operazioni.

- **GRANT assegna permessi a utenti o ruoli, consentendo loro operazioni specifiche su determinati oggetti. Per esempio, i ricercatori possono essere autorizzati a leggere i dati sugli inquinanti ma non a modificare i metadati delle stazioni.**

```
GRANT SELECT
ON AirQualityMeasurements
TO researcher_1;
```

```
GRANT SELECT, INSERT, UPDATE, DELETE
```

```
ON Stations
TO env_admin_role;
```

- **REVOKE revoca permessi concessi in precedenza. È essenziale quando cambiano le responsabilità o quando l'accesso deve essere limitato per ragioni di sicurezza o conformità normativa.**

```
REVOKE INSERT
ON AirQualityMeasurements
FROM researcher_1;
```

- **DENY vieta esplicitamente determinate azioni, prevalendo su eventuali autorizzazioni che altrimenti le consentirebbero. È una misura rigorosa per applicare le politiche di sicurezza anche in presenza di diversi livelli di permesso.**

```
DENY DELETE
ON Pollutants
TO researcher_2;
```

Le istruzioni DCL contribuiscono quindi a garantire riservatezza, integrità dei dati e collaborazione controllata tra gruppi di lavoro.

4. Linguaggio di controllo delle transazioni (TCL)

Il TCL assicura che gruppi di operazioni DML collegate siano eseguiti atomicamente: o tutte hanno successo oppure nessuna modifica parziale viene conservata. Le transazioni sono particolarmente importanti quando si gestiscono dati ambientali interconnessi, nei quali le incoerenze possono compromettere le analisi.

- **COMMIT conclude una transazione, rendendo permanenti tutte le modifiche eseguite dall'ultimo commit. Le misurazioni inserite, i metadati aggiornati o i record eliminati entrano così a far parte del dataset ufficiale.**

```
BEGIN TRANSACTION;

INSERT INTO AirQualityMeasurements (station_id, pollutant, value)
VALUES (101, 'PM2.5', 23.5);

COMMIT;
```

- ROLLBACK annulla tutte le modifiche effettuate durante la transazione corrente. Se si verifica un errore nell'aggiornamento di più tabelle, ad esempio collegando una nuova stazione a diverse tabelle di riferimento, ROLLBACK riporta il database a uno stato stabile.

```
BEGIN TRANSACTION;

UPDATE AirQualityMeasurements
SET value = -10     -- invalid value
WHERE measurement_id = 455;

-- Rilevato errore: annullare tutte le modifiche
ROLLBACK;
```

- **SAVEPOINT crea punti di ripristino denominati all'interno di una transazione, consentendo annullamenti parziali. È utile in operazioni complesse, come l'importazione di grandi lotti di dati dei sensori, quando è necessario tornare a uno stato intermedio senza ricominciare da capo.**

```
BEGIN TRANSACTION;

UPDATE Stations
SET region = 'Central Basin'
```

```
WHERE station_id = 101;

SAVEPOINT sp1;

UPDATE Stations
SET region = 'Mountain Highlands'
WHERE station_id = 102;

-- Annullare soltanto il secondo aggiornamento
ROLLBACK TO sp1;

COMMIT;
```

Attraverso il TCL, SQL garantisce coerenza e affidabilità dei dati anche durante operazioni complesse articolate in più passaggi.

Proprietà ACID delle transazioni

I dati ambientali, come misurazioni della qualità dell'aria, osservazioni meteorologiche, registri dei sensori e metadati regionali, sono spesso interconnessi e aggiornati continuamente. Se anche una sola parte viene danneggiata o aggiornata soltanto in parte, i risultati analitici e le serie storiche di monitoraggio possono essere compromessi.

Per garantire un comportamento prevedibile e coerente delle operazioni in più passaggi, i moderni database utilizzano le transazioni, ossia gruppi di operazioni da eseguire come un'unica unità. La loro correttezza è garantita da quattro principi fondamentali, noti come proprietà ACID: atomicità, consistenza, isolamento e durabilità.

1. Atomicità

L'atomicità richiede che una transazione sia trattata come un'unità indivisibile: o tutte le sue operazioni hanno successo oppure nessuna produce effetti.

Una transazione può comprendere numerosi passaggi, come inserire una misurazione, aggiornare la calibrazione di un sensore e registrare un evento. L'atomicità garantisce che il

database non si trovi mai in uno stato in cui solo alcune operazioni siano state completate. In caso di errore o guasto durante l'esecuzione, l'intera transazione viene annullata e si ritorna allo stato stabile precedente.

Esempio. Una stazione invia un lotto di misurazioni della qualità dell'aria. Il sistema deve inserirle tutte. Se una sola non può essere inserita per un errore di formato, l'atomicità garantisce che nessun record venga acquisito, evitando dati parziali e fuorvianti.

L'atomicità protegge l'integrità delle operazioni complesse ed evita aggiornamenti parziali che potrebbero alterare le analisi ambientali.

2. Consistenza

La consistenza garantisce che ogni transazione trasformi il database da uno stato valido a un altro, rispettando tutte le regole, i vincoli e le relazioni definiti.

Queste regole comprendono tipi di dato, chiavi primarie ed esterne, vincoli CHECK e regole di validazione scientifica. Un database consistente le rispetta sempre prima e dopo ogni transazione.

Esempio. Se il database impone concentrazioni di inquinanti non negative, un tentativo di inserire il valore -15 deve fallire. Il sistema respinge dati non validi per preservare la correttezza scientifica.

La consistenza garantisce che i dati ambientali non violino vincoli fisici, normativi o logici.

3. Isolamento

Nei sistemi reali, molti utenti e processi accedono contemporaneamente al database. L'isolamento assicura che l'esecuzione di una transazione non interferisca con le altre. Fino al completamento, i suoi risultati intermedi non sono visibili agli altri.

Senza isolamento, due transazioni che aggiornano gli stessi metadati di una stazione o inseriscono misurazioni sovrapposte potrebbero influenzarsi reciprocamente, producendo risultati imprevedibili.

Esempio. Due ricercatori aggiornano contemporaneamente attributi diversi della tabella Stations. L'isolamento assicura che gli aggiornamenti non entrino in conflitto, non si sovrascrivano e non generino metadati incoerenti.

La maggior parte dei DBMS supporta diversi livelli di isolamento, tra cui:

- Lettura di dati non confermati (Read Uncommitted)
- Lettura di dati confermati (Read Committed)
- Lettura ripetibile (Repeatable Read)

- Serializzabile (Serializable)

Livelli più elevati offrono garanzie maggiori ma possono ridurre la concorrenza. Nei sistemi ambientali che acquisiscono grandi lotti di dati dei sensori è fondamentale trovare il giusto equilibrio.

4. Durabilità

La durabilità garantisce che, dopo il commit, i risultati di una transazione diventino permanenti anche in caso di guasti hardware, interruzioni dell'alimentazione o arresti del sistema.

I sistemi di database garantiscono la durabilità mediante:

- registri di scrittura anticipata (write-ahead log),
- punti di controllo (checkpoint),
- archiviazione ridondante,
- meccanismi di ripristino automatico.

Esempio. Dopo aver inserito le osservazioni meteorologiche di un'intera giornata ed eseguito COMMIT, i dati devono rimanere conservati anche se subito dopo si verifica un guasto. La durabilità protegge le registrazioni di monitoraggio ambientale per anni di analisi e rendicontazione.

Le proprietà ACID, atomicità, consistenza, isolamento e durabilità, garantiscono che le operazioni sui dati siano complete, valide, indipendenti dalle interferenze e permanenti.

Caso applicativo

https://freesql.com/Per imparare SQL utilizzeremo lo strumento online Oracle FreeSQL (https://freesql.com/). Oracle FreeSQL offre agli studenti l'accesso gratuito a un vero database Oracle nel cloud, consentendo di esercitarsi con SQL e i concetti dei database relazionali in un ambiente professionale. Per utilizzarlo occorre accedere al proprio account oppure registrarsi.

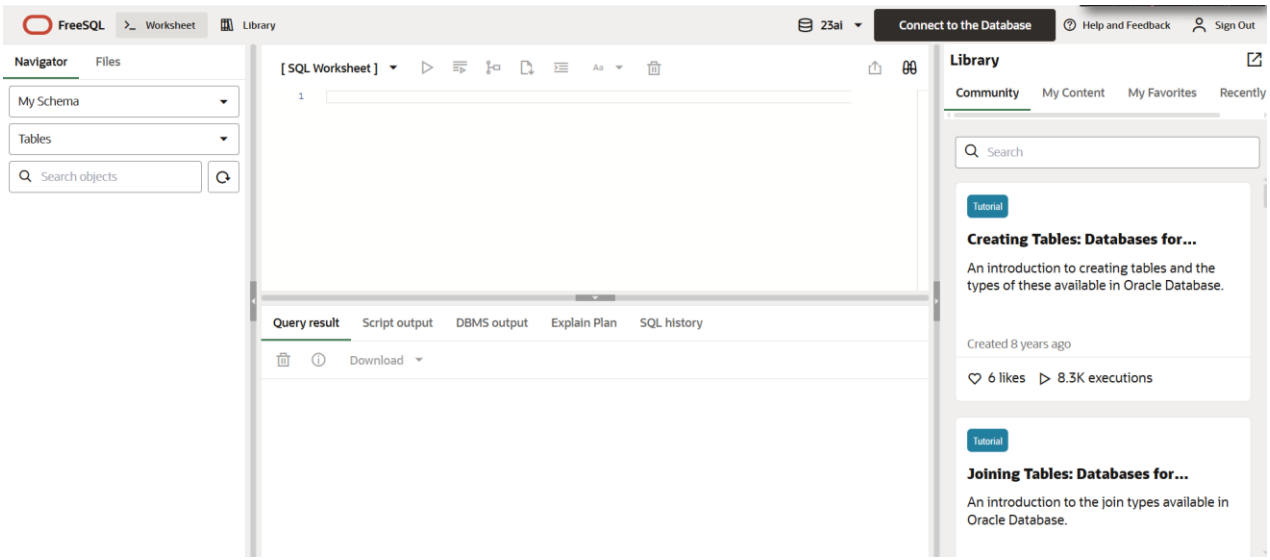


Fig. 5. Ambiente Oracle FreeSQL

Prima di esercitarci con SQL, è importante capire quale database costruiremo e per quale scopo.

In questo corso realizzeremo un database relazionale didattico che modella il monitoraggio della qualità dell'aria nella città di Varsavia, in Polonia. Il database è pensato per la formazione e si ispira alla logica dei sistemi informativi ambientali reali utilizzati da enti pubblici e istituti di ricerca. Sebbene i valori siano semplificati a fini didattici, la struttura riflette relazioni e vincoli realistici.

Il database rappresenta il modo in cui le stazioni raccolgono le misurazioni dell'inquinamento atmosferico, come vengono descritte dai metadati e come vengono definiti e regolamentati i vari inquinanti. Ogni concetto è memorizzato in una tabella distinta, secondo i principi del modello relazionale.

Al centro del database vi sono le stazioni di monitoraggio, ossia i luoghi fisici di Varsavia in cui sono installati i sensori della qualità dell'aria. Ogni stazione possiede un identificativo, un nome, una posizione geografica e attributi descrittivi come regione e gestore. Le stazioni vengono memorizzate separatamente, così da mantenere i loro metadati anche in assenza di misurazioni.

Un altro componente importante è il catalogo degli inquinanti. Sostanze come biossido di azoto (NO_2), particolato (PM10, PM2.5), ozono (O_3) e biossido di zolfo (SO_2) vengono definite una sola volta e richiamate in modo coerente nel database. Ciò evita duplicazioni, garantisce l'integrità dei dati e consente confronti significativi tra le misurazioni.

I dati delle misurazioni costituiscono la parte principale del database. Ogni misurazione registra la concentrazione di uno specifico inquinante in una data stazione e in un determinato momento.

Le misurazioni sono collegate alle stazioni e agli inquinanti tramite chiavi relazionali, consentendo interrogazioni che combinano posizione, tempo e tipo di inquinante.

Il database contiene inoltre una tabella con valori limite normativi, che rappresentano una versione semplificata dei limiti europei sulla qualità dell'aria. Questi valori consentono interrogazioni per individuare i superamenti e confrontare le concentrazioni misurate con gli standard accettati.

La struttura complessiva segue i principi della normalizzazione e dell'integrità referenziale. Ogni concetto del mondo reale è rappresentato da una propria tabella e le relazioni tra tabelle sono definite esplicitamente. Questa progettazione rende il database coerente, estensibile e adatto alle analisi SQL con join, aggregazioni e sottoquery.

1. Operazioni SQL CREATE

Creiamo un database realizzando le tabelle passo dopo passo.

1.1 Creare la tabella stations scrivendo il codice SQL nel campo SQL Worksheet, quindi premere il pulsante Run Statement:

```
CREATE TABLE stations (
  station_id     NUMBER PRIMARY KEY,
  station_name   VARCHAR2(120) NOT NULL,
  country_code   CHAR(2) DEFAULT 'PL' NOT NULL,
  city           VARCHAR2(60) DEFAULT 'Warsaw' NOT NULL,
  region         VARCHAR2(60),
  latitude       NUMBER(9,6),
  longitude      NUMBER(9,6)
);
```

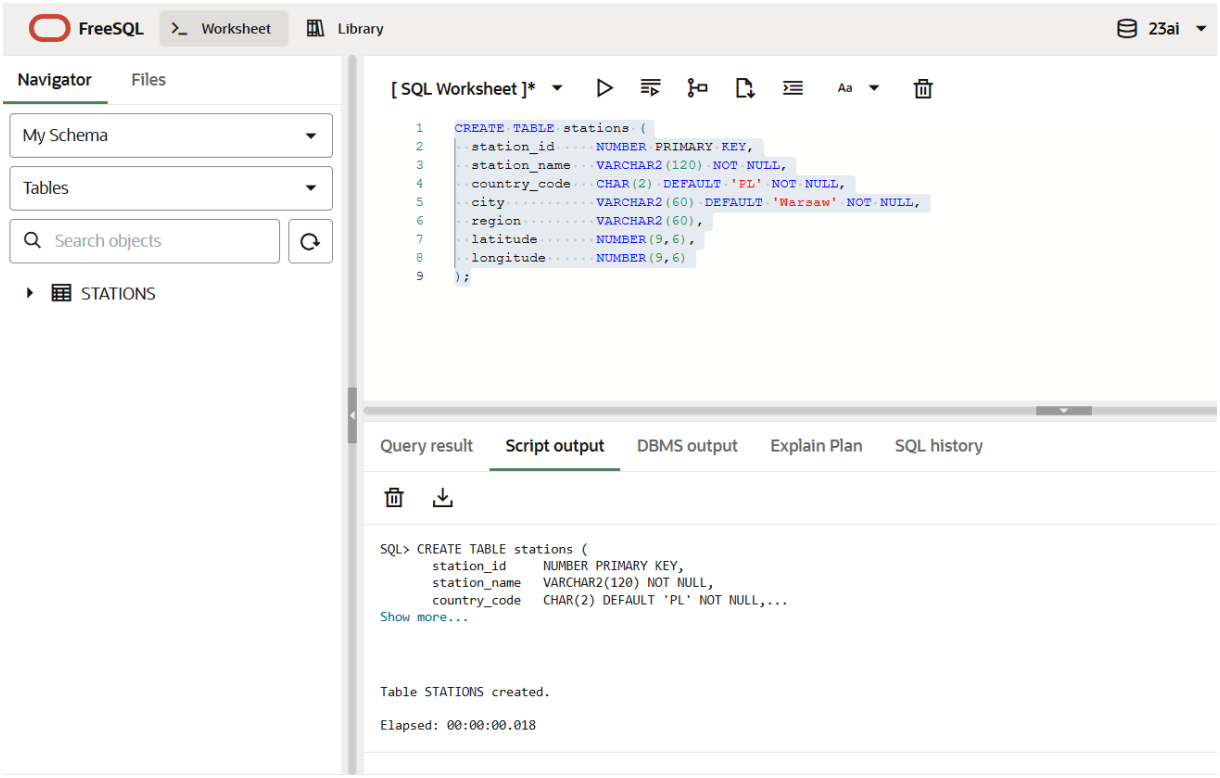


Fig. 6. Creazione della tabella stations

1.2 Creazione della tabella pollutants

```
CREATE TABLE pollutants (
  pollutant_code VARCHAR2(20) PRIMARY KEY,
  description    VARCHAR2(200) NOT NULL,
  default_unit   VARCHAR2(20) NOT NULL
);
```

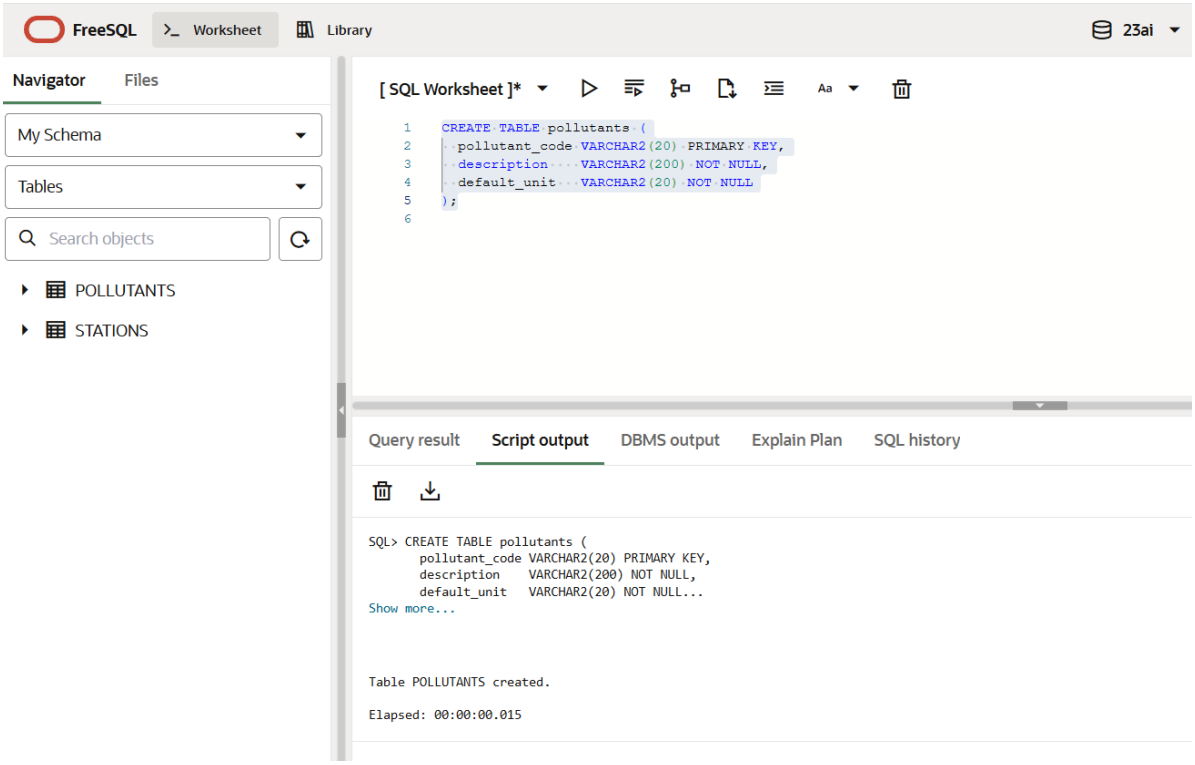


Fig. 7. Creazione della tabella pollutants

1.3 Creazione della tabella delle misurazioni della qualità dell'aria:

```
CREATE TABLE air_quality_measurements (
  measurement_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  station_id     NUMBER NOT NULL,
  pollutant_code VARCHAR2(20) NOT NULL,
  measurement_ts TIMESTAMP NOT NULL,
  value          NUMBER,
  unit           VARCHAR2(20),

  CONSTRAINT fk_station
    FOREIGN KEY (station_id)
    REFERENCES stations(station_id),

  CONSTRAINT fk_pollutant
    FOREIGN KEY (pollutant_code)
    REFERENCES pollutants(pollutant_code)
);
```

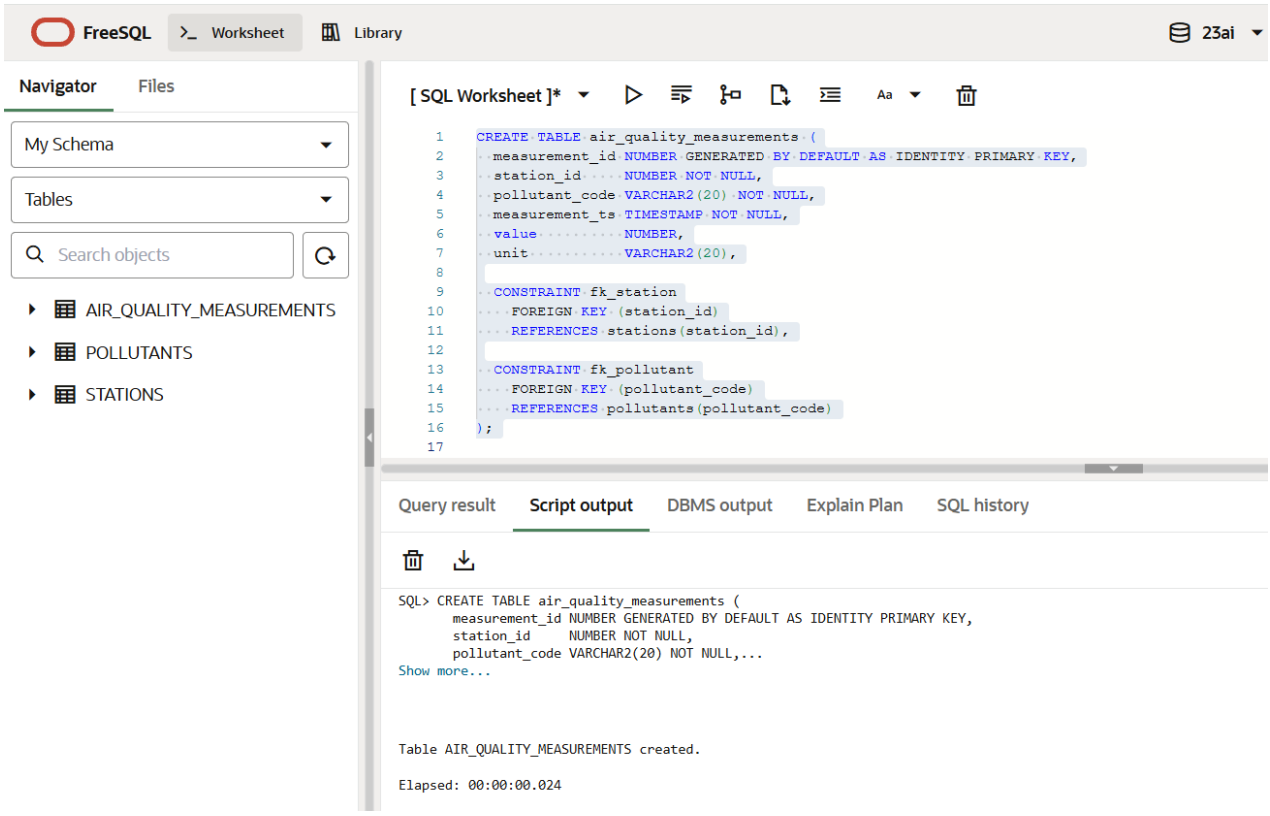


Fig. 8. Creazione della tabella pollutants

A questo punto il database è stato creato. Si può verificare visivamente la presenza delle tabelle osservando i loro nomi nel riquadro Navigator a sinistra: STATIONS, POLLUTANTS, AIR_QUALITY_MEASUREMENTS.

2. Operazioni INSERT

Ora popoliamo le tabelle con i dati.

2.1 Inseriamo i dati nella tabella stations:

```
INSERT INTO stations VALUES
(101, 'Warsaw-Center', 'PL', 'Warsaw', 'Mazowieckie', 52.229676, 21.012229);

INSERT INTO stations VALUES
(102, 'Warsaw-Traffic', 'PL', 'Warsaw', 'Mazowieckie', 52.239000, 21.005000);
```

```
INSERT INTO stations VALUES
(103, 'Warsaw-Suburb', 'PL', 'Warsaw', 'Mazowieckie', 52.300000, 20.900000);

INSERT INTO stations VALUES
(104, 'Warsaw-Park', 'PL', 'Warsaw', 'Mazowieckie', 52.260000, 21.020000);

-- Stazione senza misurazioni (per provare LEFT JOIN)
INSERT INTO stations VALUES
(105, 'Warsaw-NewStation', 'PL', 'Warsaw', 'Mazowieckie', 52.215000, 21.040000);
```

Dopo aver eseguito il comando, facendo clic con il pulsante destro sulla tabella STATIONS nel pannello di sinistra possiamo aprirla e verificare che i dati siano stati inseriti (Fig. 9).

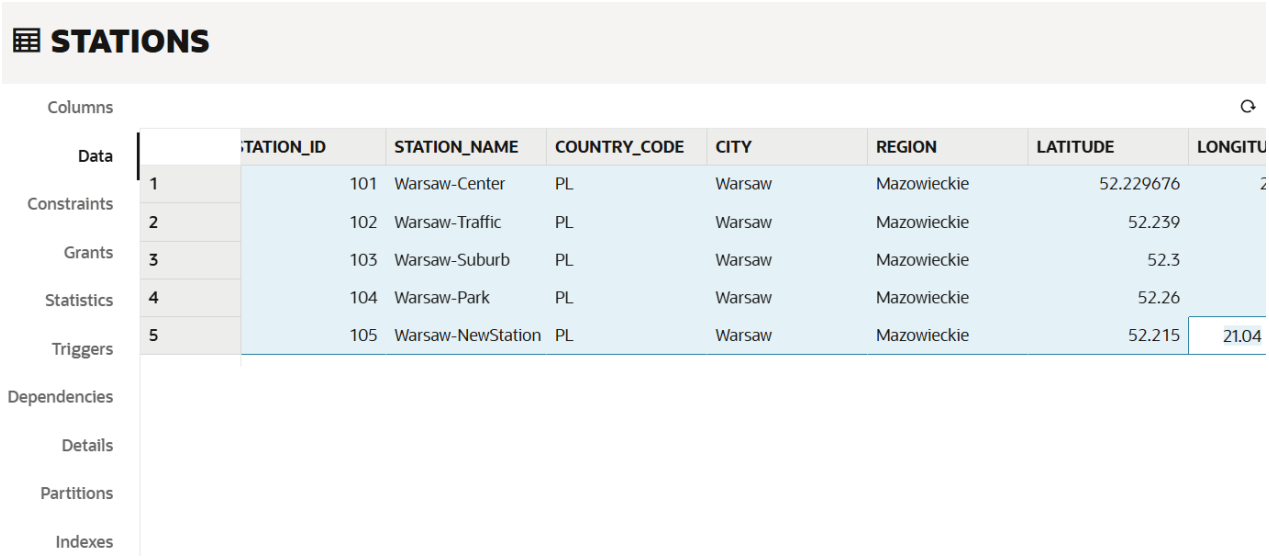


Fig. 9. Risultato dell'inserimento dei dati nella tabella stations

2.2 Inseriamo ora le informazioni sugli inquinanti:

```
INSERT INTO pollutants VALUES
```

```
('NO2', 'Nitrogen dioxide', 'ug/m3');

INSERT INTO pollutants VALUES
('PM25', 'Particulate matter < 2.5 µm', 'ug/m3');

INSERT INTO pollutants VALUES
('PM10', 'Particulate matter < 10 µm', 'ug/m3');

INSERT INTO pollutants VALUES
('O3', 'Ozone', 'ug/m3');

-- Inquinante senza misurazioni (per provare RIGHT JOIN)
INSERT INTO pollutants VALUES
('SO2', 'Sulfur dioxide', 'ug/m3');
```

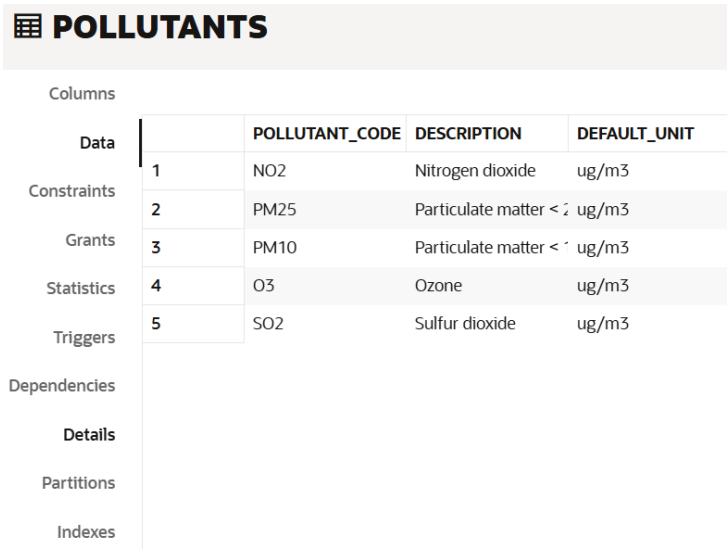


Fig. 10. Risultato dell'inserimento dei dati sugli inquinanti

2.3 Ora inseriamo i dati delle misurazioni:

```
INSERT INTO air_quality_measurements
(station_id, pollutant_code, measurement_ts, value, unit)
VALUES
(101, 'NO2', TIMESTAMP '2025-03-01 08:00:00', 45, 'ug/m3');
```

```
INSERT INTO air_quality_measurements
(station_id, pollutant_code, measurement_ts, value, unit)
VALUES
(102, 'PM25', TIMESTAMP '2025-03-01 08:00:00', 28, 'ug/m3');

INSERT INTO air_quality_measurements
(station_id, pollutant_code, measurement_ts, value, unit)
VALUES
(101, 'NO2', TIMESTAMP '2025-03-01 09:00:00', 78, 'ug/m3');

INSERT INTO air_quality_measurements
(station_id, pollutant_code, measurement_ts, value, unit)
VALUES
(102, 'NO2', TIMESTAMP '2025-03-01 08:00:00', 210, 'ug/m3');

INSERT INTO air_quality_measurements
(station_id, pollutant_code, measurement_ts, value, unit)
VALUES
(103, 'PM25', TIMESTAMP '2025-03-01 08:00:00', 18, 'ug/m3');

INSERT INTO air_quality_measurements
(station_id, pollutant_code, measurement_ts, value, unit)
VALUES
(104, 'PM10', TIMESTAMP '2025-03-01 08:00:00', 62, 'ug/m3');

INSERT INTO air_quality_measurements
(station_id, pollutant_code, measurement_ts, value, unit)
VALUES
(104, 'O3', TIMESTAMP '2025-03-01 13:00:00', 130, 'ug/m3');

-- Misurazione errata (per provare UPDATE e DELETE)
INSERT INTO air_quality_measurements
(station_id, pollutant_code, measurement_ts, value, unit)
```

```
VALUES
(101, 'PM25', TIMESTAMP '2025-03-01 10:00:00', -7, 'ug/m3');
```

AIR_QUALITY_MEASUREMENTS

Columns | Data | Constraints | Grants | Statistics | Triggers | Dependencies | Details | Partitions | Indexes

	MEASUREMENT_ID	STATION_ID	POLLUTANT_CODE	MEASUREMENT_TS	VALUE	UNIT
1	1	101	NO2	2025-03-01T08:00:(	45	ug/m3
2	2	102	PM25	2025-03-01T08:00:(	28	ug/m3
3	3	101	NO2	2025-03-01T09:00:(	78	ug/m3
4	4	102	NO2	2025-03-01T08:00:(	210	ug/m3
5	5	103	PM25	2025-03-01T08:00:(	18	ug/m3
6	6	104	PM10	2025-03-01T08:00:(	62	ug/m3
7	7	104	O3	2025-03-01T13:00:(	130	ug/m3
8	8	101	PM25	2025-03-01T10:00:(	-7	ug/m3

Fig. 11. Risultato dell'inserimento dei dati delle misurazioni

3. Operazioni UPDATE e DELETE

3.2 Con UPDATE possiamo correggere i valori errati, sostituendo quelli negativi con NULL; poi possiamo utilizzare DELETE per eliminare tutte le voci non valide:

```
UPDATE air_quality_measurements
SET value = NULL
WHERE value < 0;
DELETE FROM air_quality_measurements
WHERE value IS NULL;
```

AIR_QUALITY_MEASUREMENTS

Columns | Data | Constraints | Grants | Statistics | Triggers | Dependencies | Details

	MEASUREMENT_ID	STATION_ID	POLLUTANT_CODE	MEASUREMENT_TS	VALUE	UNIT
1	1	101	NO2	2025-03-01T08:00:(	45	ug/m3
2	2	102	PM25	2025-03-01T08:00:(	28	ug/m3
3	3	101	NO2	2025-03-01T09:00:(	78	ug/m3
4	4	102	NO2	2025-03-01T08:00:(	210	ug/m3
5	5	103	PM25	2025-03-01T08:00:(	18	ug/m3
6	6	104	PM10	2025-03-01T08:00:(	62	ug/m3
7	7	104	O3	2025-03-01T13:00:(	130	ug/m3

Fig. 12. Risultato della rimozione dei valori errati

3.2 Scrivere un comando analogo per eliminare le stazioni il cui attributo region è NULL.

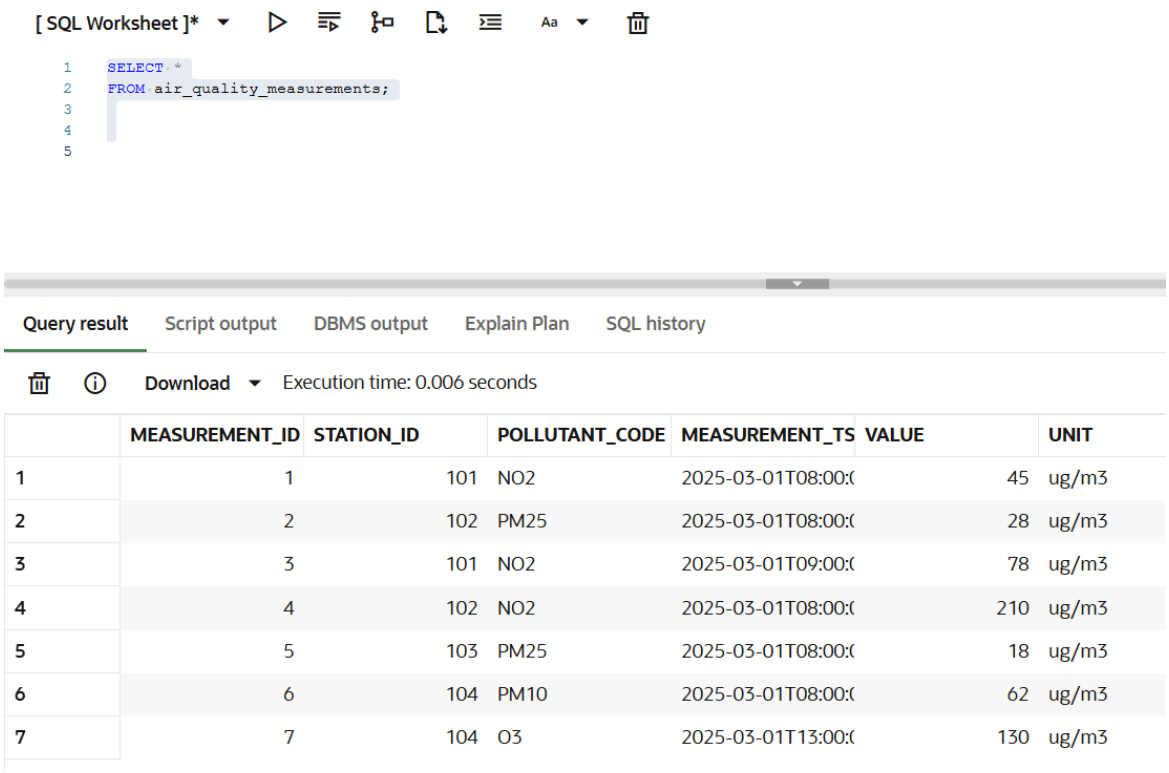
4. Interrogazioni SELECT

Abbiamo studiato i comandi DML: proviamo ora l'istruzione SELECT.

4.1 SELECT semplice:

```
SELECT *

FROM air_quality_measurements;
```



	MEASUREMENT_ID	STATION_ID	POLLUTANT_CODE	MEASUREMENT_TS	VALUE	UNIT
1	1	101	NO2	2025-03-01T08:00:(	45	ug/m3
2	2	102	PM25	2025-03-01T08:00:(	28	ug/m3
3	3	101	NO2	2025-03-01T09:00:(	78	ug/m3
4	4	102	NO2	2025-03-01T08:00:(	210	ug/m3
5	5	103	PM25	2025-03-01T08:00:(	18	ug/m3
6	6	104	PM10	2025-03-01T08:00:(	62	ug/m3
7	7	104	O3	2025-03-01T13:00:(	130	ug/m3

Fig. 13. Risultato dell'interrogazione

4.2 Verifichiamo in pratica come cambia il risultato usando ORDER BY per ordinare i dati:

```
SELECT station_id, pollutant_code, value
FROM air_quality_measurements
ORDER BY value DESC;
```

4.3 Usiamo ora WHERE e gli operatori logici per ottenere soltanto i dati relativi all'inquinante specificato:

```
SELECT station_id, pollutant_code, value
FROM air_quality_measurements
WHERE value > 50
  AND pollutant_code = 'NO2';
```

Query result | Script output | DBMS output | Explain Plan | SQL history

Download | Execution time: 0.007 seconds

	STATION_ID	POLLUTANT_CODE	VALUE
1	101	NO2	78
2	102	NO2	210

Fig. 14. Risultato dell'interrogazione

4.4 Proviamo ad applicare le funzioni di aggregazione con raggruppamento. Come cambia l'output?

```
SELECT station_id, pollutant_code, AVG(value) AS avg_value
FROM air_quality_measurements
WHERE value IS NOT NULL
GROUP BY station_id, pollutant_code;
```

4.5 Per filtrare gli inquinanti con valori medi superiori a 40, usiamo l'aggregazione con HAVING:

```
SELECT station_id, pollutant_code, AVG(value) AS avg_value
FROM air_quality_measurements
WHERE value IS NOT NULL
GROUP BY station_id, pollutant_code
HAVING AVG(value) > 40;
```

Query result | Script output | DBMS output | Explain Plan | SQL history

Download | Execution time: 0.073 seconds

	STATION_ID	POLLUTANT_CODE	AVG_VALUE
1	101	NO2	61.5
2	102	NO2	210
3	104	PM10	62
4	104	O3	130

Fig. 15. Risultato dell'interrogazione

5. Operazioni JOIN

In un database relazionale le informazioni raramente si trovano in un'unica tabella. Sono invece distribuite in più tabelle, ognuna delle quali rappresenta un concetto del mondo reale. Per analizzarle efficacemente, spesso occorre combinare righe di tabelle diverse in un solo insieme di risultati. A questo serve l'operazione JOIN di SQL.

Un JOIN definisce come collegare le righe di due tabelle mediante una condizione logica, di norma tra la chiave primaria di una tabella e la chiave esterna dell'altra. Nel database didattico del corso, le misurazioni sono collegate alle stazioni mediante station_id. Così le interrogazioni possono recuperare contemporaneamente valori misurati e informazioni descrittive delle stazioni.

5.1 INNER JOIN restituisce soltanto le righe per cui esistono valori corrispondenti in entrambe le tabelle. Le righe prive di corrispondenza vengono escluse.

Nel database della qualità dell'aria, un INNER JOIN tra la tabella delle misurazioni e quella delle stazioni restituisce solo misurazioni associate a una stazione conosciuta. È il tipo di join più comune ed è adatto quando occorre ignorare dati incompleti o non collegati.

Concettualmente, INNER JOIN risponde alla domanda: «Quali record hanno una corrispondenza in entrambe le tabelle?»

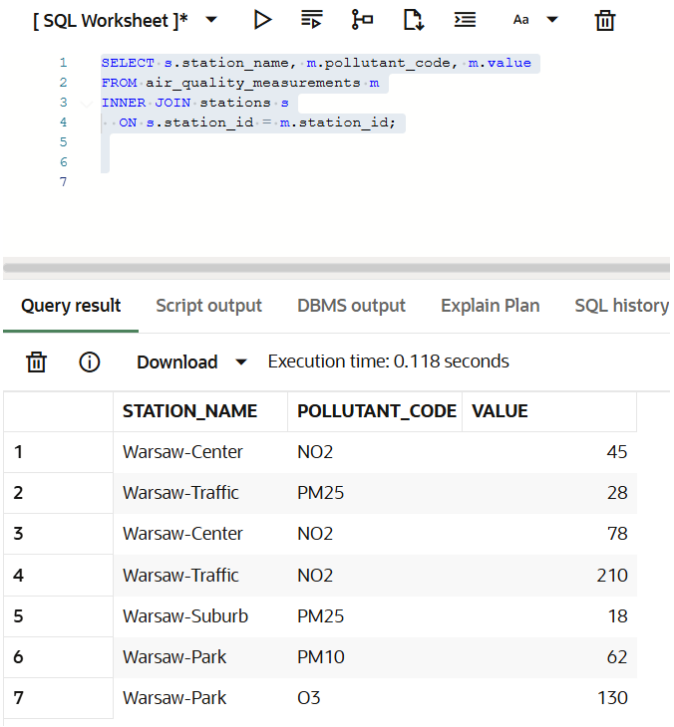
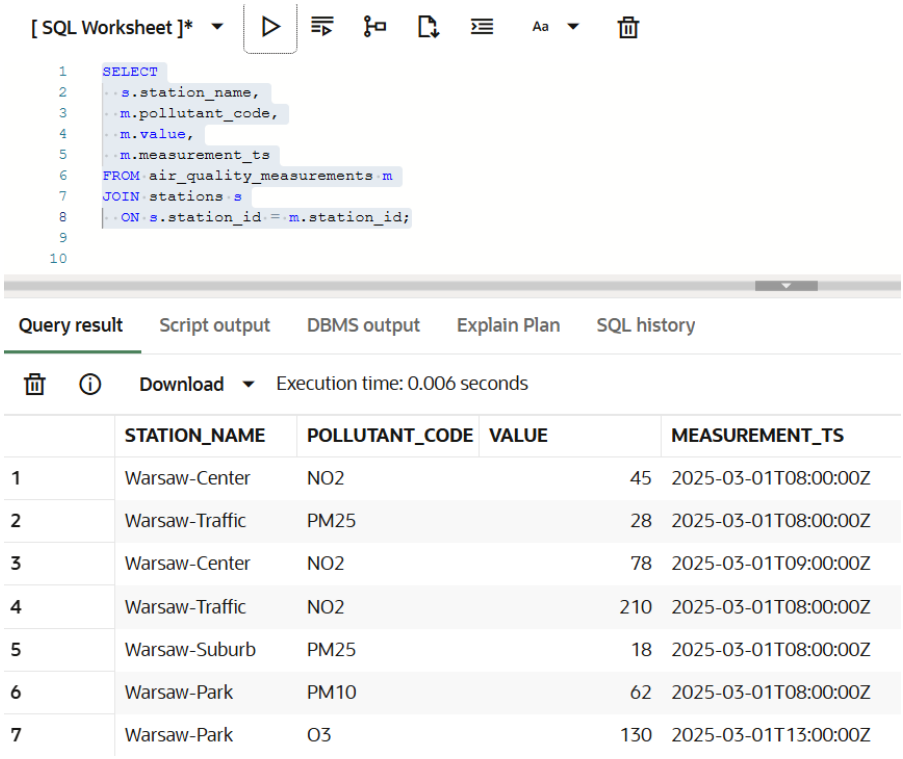


Fig. 16. Risultato dell'interrogazione INNER JOIN

Anche quando nell'interrogazione compare soltanto JOIN, si intende INNER JOIN (Fig. 17).



[SQL Worksheet]*

```
SELECT
  s.station_name,
  m.pollutant_code,
  m.value,
  m.measurement_ts
FROM air_quality_measurements m
JOIN stations s
  ON s.station_id = m.station_id;
```

Query result | Script output | DBMS output | Explain Plan | SQL history

Download | Execution time: 0.006 seconds

	STATION_NAME	POLLUTANT_CODE	VALUE	MEASUREMENT_TS
1	Warsaw-Center	NO2	45	2025-03-01T08:00:00Z
2	Warsaw-Traffic	PM25	28	2025-03-01T08:00:00Z
3	Warsaw-Center	NO2	78	2025-03-01T09:00:00Z
4	Warsaw-Traffic	NO2	210	2025-03-01T08:00:00Z
5	Warsaw-Suburb	PM25	18	2025-03-01T08:00:00Z
6	Warsaw-Park	PM10	62	2025-03-01T08:00:00Z
7	Warsaw-Park	O3	130	2025-03-01T13:00:00Z

Fig. 17. Risultato dell'interrogazione INNER JOIN

5.2 LEFT JOIN restituisce tutte le righe della tabella di sinistra, anche in assenza di una riga corrispondente nella tabella di destra. In tal caso, le colonne della tabella di destra contengono NULL.

Questo join è utile per individuare dati mancanti o incompleti. Ad esempio, collegando le stazioni alle misurazioni, un LEFT JOIN elenca tutte le stazioni, comprese quelle senza misurazioni registrate.

LEFT JOIN risponde alla domanda: «Quali record sono presenti nella tabella di sinistra e quali dati collegati sono disponibili, se esistono, nella tabella di destra?»

[SQL Worksheet]*

```
1  SELECT s.station_name, m.pollutant_code, m.value
2  FROM stations s
3  LEFT JOIN air_quality_measurements m
4      ON s.station_id = m.station_id
5  ORDER BY s.station_name;
6
7
8
```

Query result | Script output | DBMS output | Explain Plan | SQL history

Download | Execution time: 0.073 seconds

	STATION_NAME	POLLUTANT_CODE	VALUE
1	Warsaw-Center	NO2	45
2	Warsaw-Center	NO2	78
3	Warsaw-NewStation	*(null)*	*(null)*
4	Warsaw-Park	O3	130
5	Warsaw-Park	PM10	62
6	Warsaw-Suburb	PM25	18
7	Warsaw-Traffic	PM25	28
8	Warsaw-Traffic	NO2	210

Fig. 18. Risultato dell'interrogazione LEFT JOIN

Come si vede, la stazione Warsaw-NewStation non ha misurazioni associate, ma compare comunque nei risultati.

5.3 RIGHT JOIN funziona in modo opposto: restituisce tutte le righe della tabella di destra, anche quando non c'è corrispondenza nella tabella di sinistra. In assenza di corrispondenza, le colonne della tabella di sinistra contengono NULL.

Nel database didattico, RIGHT JOIN può elencare tutti gli inquinanti definiti, compresi quelli per cui non sono ancora state registrate misurazioni. È utile per verificare la completezza dei dati di riferimento o individuare categorie inutilizzate.

RIGHT JOIN risponde alla domanda: «Quali record sono presenti nella tabella di destra e quali sono attualmente referenziati nella tabella di sinistra?»

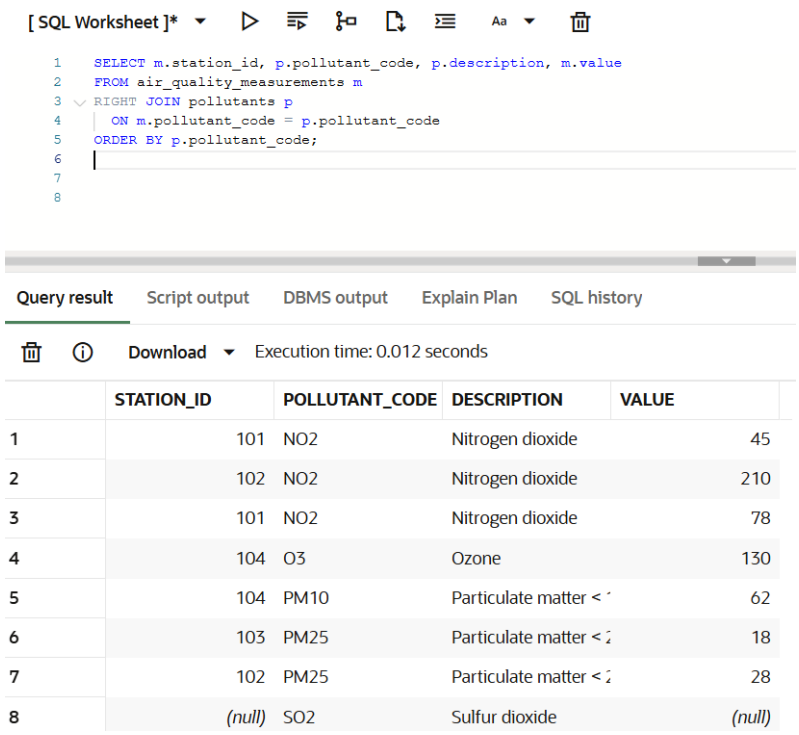


Fig. 19. Risultato dell'interrogazione RIGHT JOIN

Come si vede, l'ultima riga dei risultati di RIGHT JOIN mostra SO2, che non è associato ad alcuna stazione.

6. Sottoquery in SQL

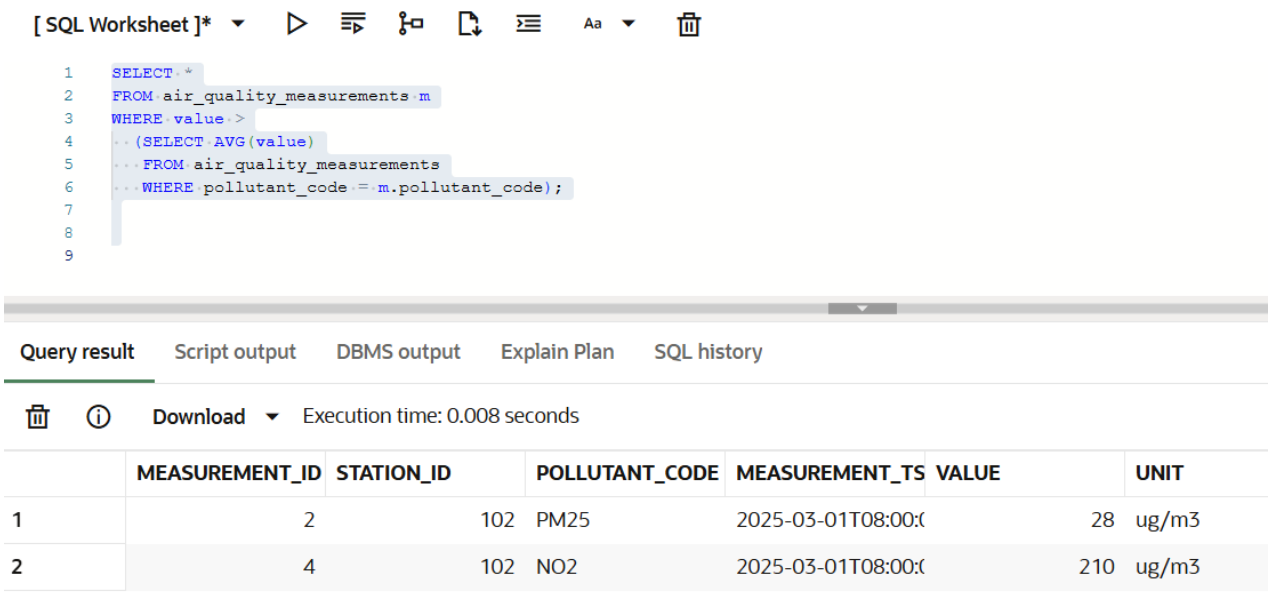
Una sottoquery è un'interrogazione annidata in un'altra istruzione SQL. Si usa quando il risultato di una query serve a determinare il risultato di un'altra. Le sottoquery permettono di esprimere logiche complesse in maniera chiara e strutturata.

Di norma, la sottoquery viene eseguita prima e il suo risultato è utilizzato dalla query esterna. È particolarmente utile nelle analisi basate su valori calcolati, come medie, massimi, minimi o soglie definite dinamicamente.

Una sottoquery può restituire un solo valore, una riga o un insieme di righe. Un utilizzo comune consiste nel confrontare singoli record con valori aggregati ricavati dalla stessa tabella.

Le sottoquery possono essere non correlate, cioè indipendenti dalla query esterna, oppure correlate, cioè dipendenti dai valori della query esterna e valutate ripetutamente per ogni riga. Benché più avanzate, anche le sottoquery correlate sono importanti nell'analisi dei dati relazionali.

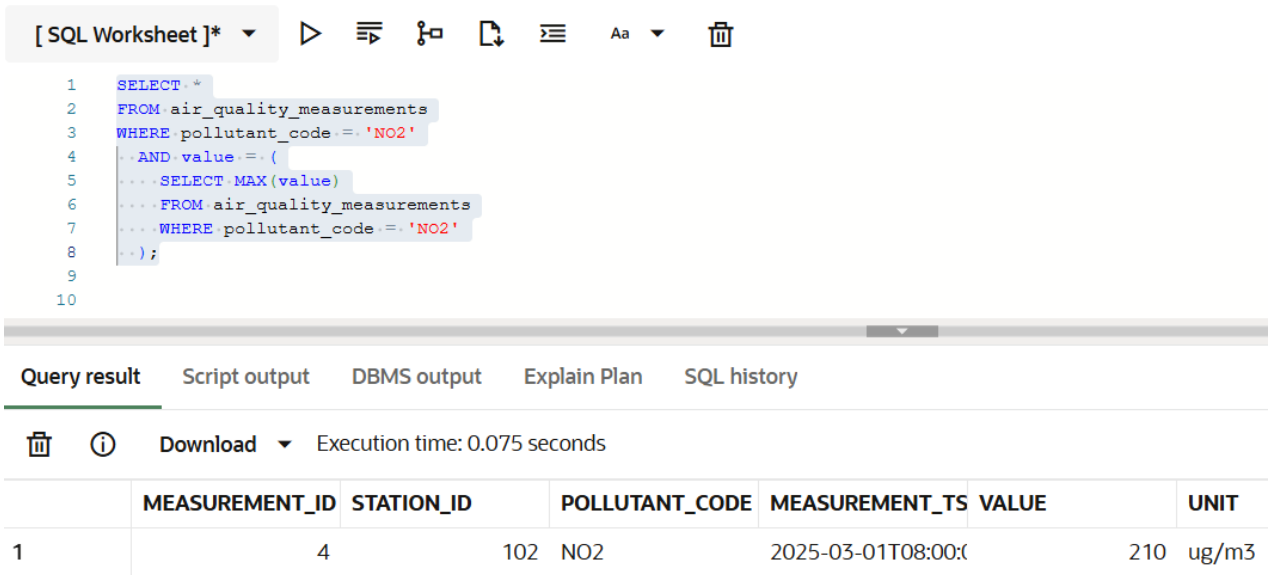
6.1 Creiamo una sottoquery che visualizzi le misurazioni superiori alla media di uno specifico inquinante (Fig. 20).



	MEASUREMENT_ID	STATION_ID	POLLUTANT_CODE	MEASUREMENT_TS	VALUE	UNIT
1	2	102	PM25	2025-03-01T08:00:(	28	ug/m3
2	4	102	NO2	2025-03-01T08:00:(	210	ug/m3

Fig. 20. Risultato dell'esecuzione della sottoquery

6.2 Una sottoquery può anche individuare le stazioni con i valori massimi di NO_2 (Fig. 21).



	MEASUREMENT_ID	STATION_ID	POLLUTANT_CODE	MEASUREMENT_TS	VALUE	UNIT
1	4	102	NO2	2025-03-01T08:00:(	210	ug/m3

Fig. 21. Risultato dell'esecuzione della sottoquery

7. Linguaggio di controllo delle transazioni (TCL)

Il TCL assicura che gruppi di operazioni SQL siano eseguiti in modo affidabile. In Oracle le transazioni si avviano automaticamente, ma devono essere concluse o annullate esplicitamente

mediante COMMIT o ROLLBACK. Il comando SAVEPOINT consente un annullamento parziale all'interno di una transazione.

7.1 Inserire una nuova misurazione della qualità dell'aria e salvare definitivamente la modifica.

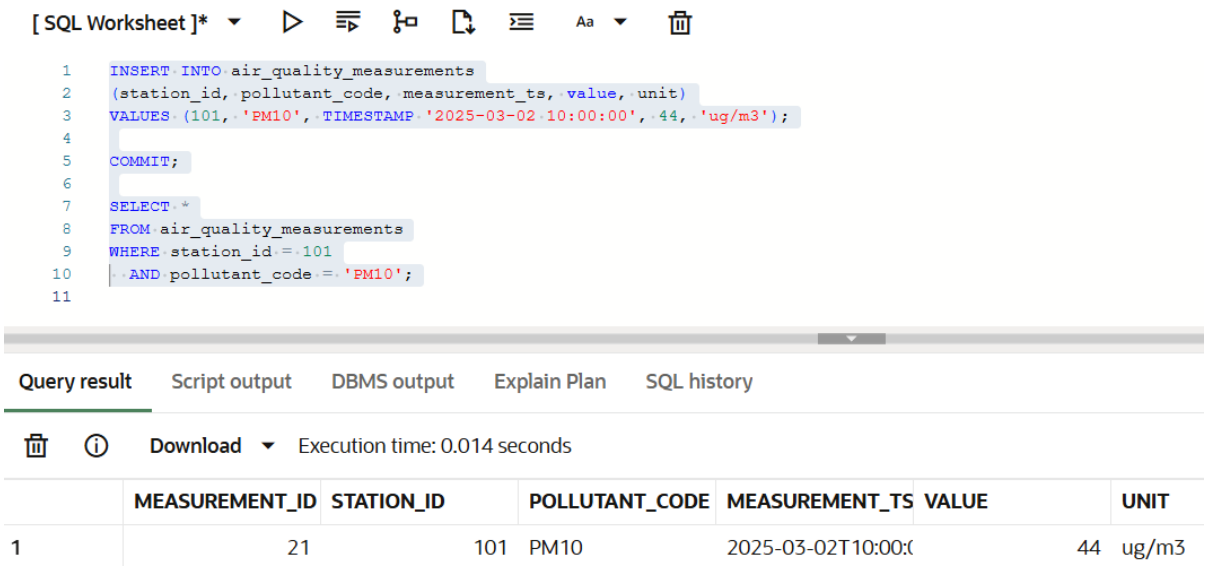


Fig. 22. Verifica della transazione confermata

7.2 Inserire una misurazione e poi annullare l'operazione con ROLLBACK.

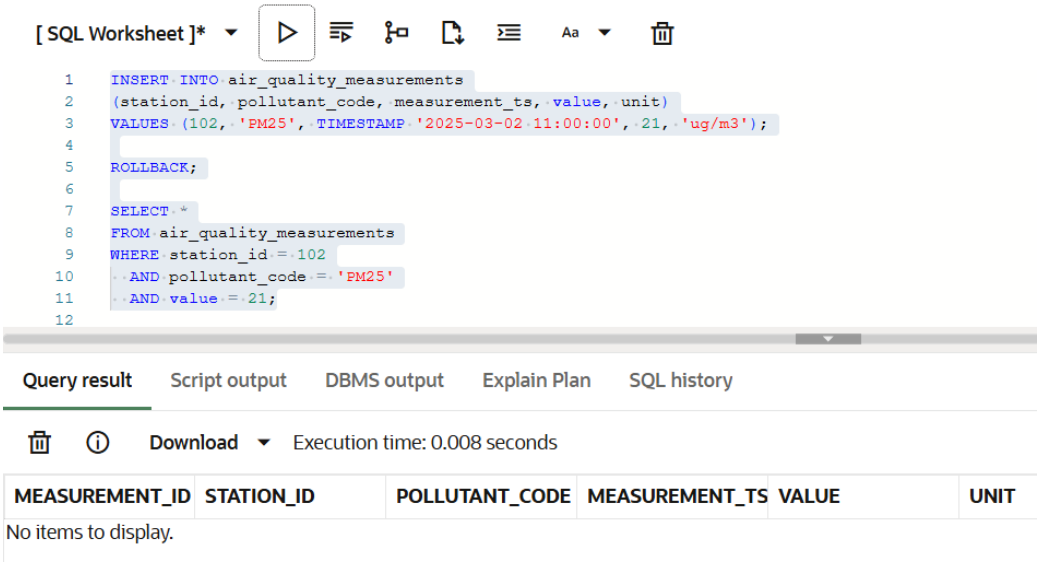


Fig. 23. Verifica dell'annullamento della transazione

7.3 Eseguire più aggiornamenti e annullare soltanto una parte della transazione.

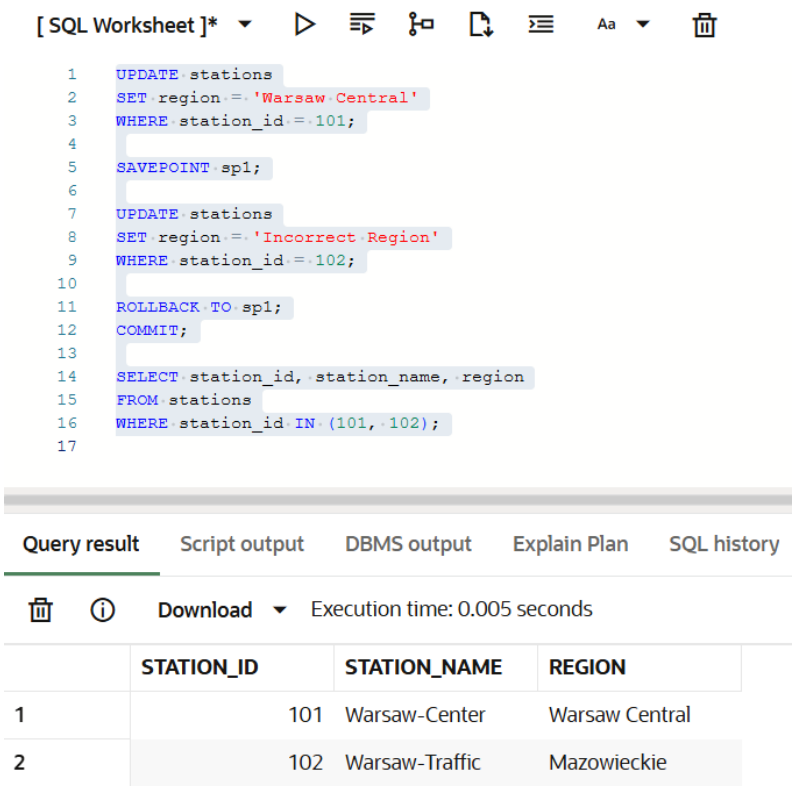


Fig. 24. Verifica dell'annullamento parziale della transazione

Attività di analisi con SQL:

1. Visualizzare tutte le misurazioni della qualità dell'aria, indicando identificativo della stazione, codice dell'inquinante, valore misurato e marca temporale. Ordinare i risultati per ora di misurazione.
2. Elencare tutte le misurazioni il cui valore è superiore a 50, mostrando identificativo della stazione, inquinante, valore e ora.
3. Recuperare tutte le misurazioni di NO_2, ordinate dal valore più alto al più basso.
4. Per ogni inquinante, individuare la stazione in cui è stato registrato il valore massimo. Mostrare nome della stazione, inquinante e valore misurato.
5. Mostrare soltanto le combinazioni stazione-inquinante per le quali il valore medio supera 40.
6. Per ciascun inquinante, trovare il valore massimo registrato.

Esercitazioni pratiche

Esercizio 1 – Creare un nuovo schema di database

Per analizzare il dataset India Urban Noise Monitoring (disponibile su https://www.kaggle.com/datasets/rohanrao/noise-monitoring-data-in-india), gli studenti devono implementare lo schema del database mediante istruzioni DDL. Creare due tabelle relazionali: STATIONS (Station, Name, City, State, Type) e STATION_MONTH (Station, Year, Month, Day, Night, DayLimit, NightLimit). Definire Station come chiave primaria della tabella STATIONS. Definire quindi una chiave esterna in STATION_MONTH, in modo che STATION_MONTH.Station faccia riferimento a STATIONS(Station), garantendo l'integrità referenziale tra le misurazioni e i metadati delle stazioni.

Esercizio 2 – Inserire dati di esempio

*stations_insert.sqlstation_month_insert.sql*Poiché Oracle FreeSQL non supporta l'importazione diretta di CSV e in questo ambiente i dati vengono caricati principalmente con INSERT, il dataset è stato convertito negli script SQL stations_insert.sql e station_month_insert.sql. Eseguire gli script per popolare le rispettive tabelle. Inserire prima i metadati delle stazioni (STATIONS), quindi le misurazioni mensili (STATION_MONTH).

Esercizio 3 – Interrogazioni di base

Scrivere semplici interrogazioni SELECT per esplorare i dati. Elencare prima tutte le stazioni di una città scelta, ad esempio Delhi. Visualizzare quindi le registrazioni mensili del rumore per una stazione e un anno selezionati. Queste interrogazioni aiutano a comprendere struttura e significato delle due tabelle prima di passare alle analisi.

Esercizio 4 – Filtri con operatori logici

Recuperare e confrontare due categorie di superamento dei limiti. Prima estrarre tutti i record in cui entrambi i limiti sono superati, ossia Day > DayLimit AND Night > NightLimit. Poi estrarre quelli in cui è superato almeno un limite, ossia Day > DayLimit OR Night > NightLimit. Interpretare come la scelta dell'operatore logico modifica il significato e la dimensione dei risultati.

Esercizio 5 – Aggregazioni

Calcolare un riepilogo per stazione, determinando il livello medio di rumore diurno e quello notturno. Raggruppare i risultati per Station. L'esercizio rafforza l'uso delle funzioni di aggregazione e di GROUP BY per la produzione di report analitici.

Esercizio 5 – Interrogazioni JOIN

Produrre un rapporto che combini i metadati delle stazioni con le misurazioni mensili. Usando un INNER JOIN tra STATIONS e STATION_MONTH, restituire nome della stazione, città, tipologia, anno, mese, rumore diurno e notturno. L'esercizio mostra come i collegamenti relazionali consentano di creare report significativi combinando dati descrittivi e misurati.

Valutazione

Quiz (5 domande): breve verifica dei concetti prima dell'attività pratica.

1. Quale affermazione descrive meglio un LEFT JOIN?

A. Restituisce soltanto le righe corrispondenti in entrambe le tabelle.

B. Restituisce tutte le righe della tabella di sinistra e le righe corrispondenti della tabella di destra.

C. Restituisce tutte le righe della tabella di destra e le righe corrispondenti della tabella di sinistra.

D. Combina tutte le righe delle due tabelle senza condizioni.

2. Quale anomalia si verifica quando eliminare l'ultima misurazione di una stazione comporta la perdita dei suoi metadati?

A. Anomalia di aggiornamento

B. Anomalia di inserimento

C. Anomalia di cancellazione

D. Anomalia di join

3. Quale comando SQL appartiene al linguaggio di definizione dei dati (DDL)?

A. SELECT

B. INSERT

C. DROP

D. UPDATE

4. Quale dei seguenti è un uso valido di una sottoquery SQL?

A. Calcolare il valore medio di un inquinante all'interno di una clausola WHERE

B. Eliminare un database

C. Concedere autorizzazioni

D. Creare nuove tabelle

5. Quale clausola viene usata per combinare le righe di due tabelle sulla base di una condizione di corrispondenza?

A. GROUP BY

B. ORDER BY

C. JOIN … ON

D. HAVING

Criteri di valutazione: correttezza delle interrogazioni SQL, completezza dei risultati, chiarezza dell'interpretazione e punteggio del quiz.

Component	Weight	Criteria
Quiz	20%	Comprensione dei concetti
Esercizi SQL	50%	Correttezza, efficienza e completezza
Relazione	30%	Interpretazione, chiarezza e visualizzazione

Riferimenti e letture di approfondimento

- Risorse cloud di Oracle Academy
- Documentazione sugli standard europei della qualità dell'aria
- Documentazione dell'API Open-Meteo
- Libri ad accesso aperto selezionati sui data warehouse
- Articoli scientifici ad accesso aperto selezionati sui data warehouse nelle scienze ambientali
- C. J. Date. Introduction to Database Systems. Addison-Wesley Longman, Inc., 2004. 983 p.
- H. Garcia-Molina, J. Ullman, J. Widom. Database Systems: The Complete Book. Pearson, 2008. 1248 p.
- P. Rob, C. Coronel. Database Systems: Design, Implementation, and Management. Course Technology, 2011. 728 p.



Co-funded by
the European Union

EcoMinds