Il linguaggio SQL è un linguaggio standard per la definizione, manipolazione e interrogazione delle basi di dati relazionali ed ha le seguenti caratteristiche:

- è dichiarativo;
- opera su multiset di tuple, ossia su insiemi di tuple che possono contenere duplicati;
- consente di tradurre tutte le interrogazioni esprimibili in algebra relazionale;

L'SQL mette a disposizione 6 tipi di dati:

1. Caratteri:
   - Char (x) --> caratteri a lunghezza fissa stabilita da x;
   - Varchar (x) --> caratteri a lunghezza variabile, con lunghezza max stabilita da x;
2. Bit: è i Booleano, che consente che assume valori 0 o 1 (vero o falso);
3. Numerico esatto:
   - Interi: Integer (o semplicemente Int) e Smallint;
   - Decimali: Decimal, dove bisogna specificare anche la precisione e la scala;
4. Numerico Approssimato:
   - Float dove bisogna specificare la precisione;
   - Real;
   - Double Precision;
5. Datatime:
   - Date: AAAA-MM-GG
   - Time: OO-MM-SS
   - TimeStamp: AAAA-MM-GG OO-MM-SS
6. Intervallo: Interval, nel quale bisogna indicare il campo iniziale e finale

   ES: Interval year (2) to month

   Il valore 2 indica il numero massimo di campi consentiti in year, ossia questo intervallo varia da 0 anni e 0 mesi, fino a 99 anni e 11 mesi

   ES: Interval year (5) to month

   Il valore 5 indica il numero massimo di campi consentiti in year, ossia questo intervallo varia da 0 anni e 0 mesi, fino a 99999 anni e 11 mesi

Per consentire i valori nulli, l'SQL utilizza una logica a 3 valori basata sulle seguenti tabelle:

AND	***true***	***null***	***false***
true	true	null	false
null	null	null	false
false	false	false	false

OR	***true***	***null***	***false***
true	true	true	true
null	true	null	null
false	true	null	false

NOT	
true	false
null	null
false	true

dove:

- un qualsiasi confronto con il valore null restituisce ancora nulli (ES: X <op> null = null [dove <op> è un operatore algebrico o relazionale e X è una costante o una variabile]);
- due valori null sono considerati diversi;
- due valori null sono considerati uguali solo ai fini dell'indicizzazione e ai fini del raggruppamento;

Schema: è una collezione di oggetti del database (tabelle, domini, indici, viste, ecc...). Tutti gli oggetti dello schema hanno lo stesso proprietario.

Sintassi:
CREATE SCHEMA [<nome_schema>]
[AUTHORIZATION <nome_proprietario>]
Dove:

- se AUTHORIZATION <nome_proprietario> viene omesso, si considera come proprietario colui che ha dato il comando;
- se <nome_schema> viene omesso, si assume come nome dello schema, quello del proprietario;
- la dichiarazione degli oggetti dello schema può avvenire al di fuori del comando CREATE SCHEMA;

Tabella: è costituita da una collezione ordinata (lista) di uno o più attributi (colonne) e da un insieme di zero o più vincoli.

Sintassi:
CREATE TABLE <nome_tabella>
(<nome_colonna> <dominio> [<vincolo_di_colonna>],
...,
<nome_colonna> <dominio> [<vincolo_di_colonna>],
[<vincolo_di_tabella>])

Domini: è un insieme di valori consentiti. Un dominio può essere di tipo di base del SQL oppure può essere definito dall'utente.

Sintassi:
CREATE DOMAIN <nome_dominio> AS <tipo_di_dati>
[<valore_default>] [<vincolo_di_dominio>]

Indici: i comandi di definizione degli indici, non fanno parte dello standard del linguaggio

Sintassi generalmente accettata:
CREATE [UNIQUE] INDEX <nome_indice>
ON <tabella> (<lista_colonne>)
NB: inserendo UNIQUE dico che non sono consentiti valori duplicati degli attributi specificati in <lista_colonne>, ossia è come se dichiarassi <lista_colonne> come chiave di <tabella>

Rimozione: è possibile rimuovere le tabelle, e gli schemi creati.

Sintassi:
DROP SCHEMA <nome_schema>
DROP TABLE <nome_tabella>

Vincoli di Tabella: permettono di controllare i valori che possono essere registrati in una tabella.

- Primary key: esprime la chiave primaria della tabella e può essere definita solo una volta nella tabella;
Sintassi:
PRIMARY KEY (<lista_colonne>)

- Foreign key:
Sintassi:
FOREIGN KEY (<lista_colonne>)
REFERENCES <tabella> [(<lista_colonne>)]
Dove:

- (<lista_colonne>) deve essere definita in <tabella> come chiave
- in assenza di (<lista_colonne>) è riferita alla primary key di <tabella>

- Check: esprime un generico vincolo sulla tabella tramite un'espressione che deve essere vera per tutte le tuple della tabella;
Sintassi:
CHECK (<condizione>)

Vincoli di Colonna: permettono di controllare i valori che possono essere registrati in una singola colonna. Tutti i vincoli di colonna possono essere espressi come vincoli di tabella.

- *Not Null*: l'attributo non può assumere il valore null;
- *Unique*: esprime l'unicità dell'attributo. (Not Null Unique dichiara che l'attributo è una chiave della tabella);
- ***Primary key***: stabilisce che l'attributo è la chiave primaria della tabella;
- *References*: esprime il vincolo delle FK, la colonna nella tabella riferita può essere la primary key oppure una colonna esplicitamente indicata;
- ***Check***: esprime un generico vincolo sulla colonna tramite un'espressione logico-relazionale;

Violazione di vincoli di integrità: quando il sistema rileva una violazione di vincoli di integrità, il comando di aggiornamento viene rifiutato e viene segnalato l'errore all'utente.

Sintassi:
FOREIGN KEY (<lista_colonne>)
REFERENCES <tabella> [(<lista_colonne>)]
ON <DELETE | UPDATE>
<CASCADE | SET NULL | SET DEFAULT | NO ACTION>

Una operazione sulla tabella che effettua il riconoscimento, detta *dipendente*, che viola il vincolo viene rifiutata. Ad una operazione sulla tabella riferita che viola il vincolo, si può rispondere con:

Cascade: in caso di modifica, il nuovo valore dell'attributo della tabella riferita viene riportato su tutte le corrispondenti righe della tabella dipendente. In caso di cancellazione, tutte le righe della tabella dipendente corrispondenti alla riga cancellata vengono cancellate.

Set Null: all'attributo referente viene assegnato il valore null al posto del valore modificato/cancellato nella tabella riferita.

Set Default: all'attributo referente viene assegnato il valore di default al posto del valore modificato/cancellato nella tabella riferita.

No Action: non viene eseguita alcuna reazione.

Modifica della struttura di una tabella

Il comando ALTER TABLE <nome-tabella> <tipo-modivica> permette di modificare la struttura di una tabella. In <tipo-modivica> si possono specificare le seguenti azioni:

Aggiunta di una colonna:
ADD [COLUMN] <nome-colonna> <dominio> [<vincolo-di-colonna>]
Rimozione di una colonna:
DROP [COLUMN] <nome-colonna> RESTRICT | CASCADE
Con RESTRICT la colonna viene eliminata solo se non ci sono dipendenze da tali colonne in altre definizioni, mentre con CASCADE si forza l'eliminazione della colonna e di tutte le dipendenze nelle altre definizioni.
Aggiunta di un vincolo di tabella:
ADD <vincolo-di-tabella>
Rimozione di un vincolo di tabella:
DROP <nome-vincolo> RESTRICT | CASCADE
Con RESTRICT un vincolo di unicità non viene eliminato se è usato un vincolo di foreing key, mentre con CASCADE si forza l'eliminazione anche del vincolo di foreing key.
Imposta e il valore di default di una colonna:
ALTER [COLUMN] <nome-colonna> SET DEFAULT <valore>
Annulla il valore di default per una colonna:
ALTER [COLUMN] <nome-colonna> DROP DEFAULT

Il comando ALTRE DOMAIN <nome-dominio> <tipo-modifica> permette di modificare alcune proprietà di un dominio di valori. Le proprietà modificabili sono quelle che non influenzano i dati reali.

Interrogazioni

L'istruzione base dell'SQL per costruire interrogazioni di complessità arbitraria è lo statement SELECT
Sintassi di base:
SELECT [DISTINCT | ALL] <lista-select>
FROM <list-from>
[WHERE <condizione>]
[ORDER BY <lista-order>]

Dove: con ALL (default) non c'è l'eliminazione dei duplicati, mentre con DISTINCT gli è l'eliminazione dei duplicati, ossia non è necessaria se in SELECT inserisco una chiave primaria

1. <lista-select>
    - uno o più attributi della lista-from;
    - funzioni aggregate: COUNT, AVG, MIN, MAX, SUM;
    - costanti;
    - una generica espressione matematica che coinvolge uno o più di te gli oggetti precedenti;
    - il simbolo *: tutti gli attributi;

2. <lista-from>
    - una o più tabelle o viste;
    - operazioni di join tra una o più tabelle o viste;
3. <condizione>:
    - predicati semplici;
    - predicati di join;
    - predicati di subquery;
4. <lista-order>:
    - uno o più attributi della <lista-select>
    - ordine ascendente (default) o discendente (DESC)

Predicati semplici

Operatori relazionali: <attr> <op-rel> <cost>
Dove <op-rel> appartiene a uno dei seguenti simboli: =, < >, >, > =, <, <=
Operatore di range: <attr> BEETWIN <cost1> AND <cost2>
Operatore dei set: <attr> IN (<cost1>, …, <costN>)
Operatore di confronto stringhe: <attr> LIKE <stringa>
Dove <stringa> può contenere i caratteri speciali _ (carattere arbitrario) e % (stringa arbitraria)
Operatori quantificati: <attr> <op-rel> [ANY | ALL] (<cost1>, …, <costN>)

Confronto con le si era vuoto ():

- <attr> <op-rel> ANY () ha valore false
- <attr> <op-rel> ALL () ha valore true

Operatore di confronto con valori NULL: <attr> IS [NOT] NULL
Ordinamento del risultato: l'ordinamento deve essere fatto rispetto a uno o più elementi della <lista-select> con tale elemento può essere indicato anche riportando la sua posizione nella <lista-select>
Esempio:

```
SELECR Matr, CC, (60*Voto)/30
FROM E
WHERE CC='C1'
ORDER BY 3 DESC, Matr
```

*ossia ordina secondo la terza colonna della lista di selezione, cioè (60*Voto)/30*

Prodotto cartesiano e Join

Il **prodotto cartesiano** di due o più relazioni si ottiene riportando le relazioni nella <lista-select> della clausola FROM, senza clausola WHERE.
Il **Join** viene espresso generalmente riportando nella clausola FROM le relazioni interessate e nella clausola WHERE le condizioni di join.
Con l'SQL 92 è possibile esprimere le operazioni di join nella clausola FROM:

<tabella1> [INNER | LEFT | RIGHT | FULL] JOIN <tabella2> ON <condizione>

Il default è il join interno, quindi la parola INNER può essere commessa.

Outer-join: oltre al join interno, con lo standard SQL 92 sono stati introdotti gli operatori di outer-join:

- <tabella1> LEFT JOIN <tabella2> ON <condizione>
  mantiene le tuple le di <tabella1> per cui non esiste corrispondenza in <tabella2>
- <tabella1> RIGHT JOIN <tabella2> ON <condizione>
  mantiene le tutte le di <tabella2> per cui non esiste corrispondenza in <tabella1>
- <tabella1> FULL JOIN <tabella2> ON <condizione>
  mantiene le tutte le di <tabella1> e di <tabella2> per cui non esiste corrispondenza in <tabella2> e <tabella1> rispettivamente.

Le tuple senza corrispondenza vengono concatenate con tuple di valori null, di lunghezza opportuna.
Nella clausola FROM è possibile esprimere più di un'operazione di JOIN.
Esempio: per ogni esame con voto superiore a 24 per riportare il nome dello studente e il codice del docente del corso.

```
SELECT S.SNome, C.CD
FROM (S JOIN E ON (S.Matr=E.Matr))
JOIN C ON (E.CC=C.CC)
WHERE Voto > 24
```

Oppure

```
SELECT S.SNome, C.CD
FROM S, E, C
WHERE S.Matr = E.Matr
AND E.CC = C.CC
AND Voto > 24
```

Self-join: nel join tra una tabella e se stessa occorre necessariamente utilizzare dei sinonimi (alias) per distinguere le diverse corrispondenze della tabella.
Esempio: coppie di studenti residenti nella stessa città

```
SELECT S1.Matr, S2.Matr
FROM S S1, S S2
WHERE S1..Città = S2.Città
AND S1.Matr < S2.Matr
```

Interrogazioni innestate: viene detta innestata o nidificata se la sua condizione è formulata usando il risultato di un'altra interrogazione chiamata **subquery.**
In generale, un'interrogazione innestata viene formulata con:

- operatori quantificati: <attr> <op-rel> [ANY | ALL] (<subquery>)
  N.B.: se uso ALL devo usare sempre o >= o <=;
- operatore di set: <attr> [NOT] IN (<subquery>)
  N.B.: NOT IN è la sottrazione;
- operatore esistenziale: [NOT] EXISTS (<subquery>)

Interrogazione innestata con operatori quantificati

il confronto tra un attributo e il risultato di un'interrogazione, <attr> <op-rel> (<subquery>) non è in generale corretto in quanto si confronta il singolo valore assunto da <attr> con le insieme di valori restituiti da <subquery>. Tuttavia, il confronto è possibile quando <subquery> produce (run-time) un valore atomico. Nel confronto tra un attributo e il risultato di una interrogazione occorre specificare, dopo l'operatore relazionale <op-rel>, ANY oppure ALL.

Esempio: studenti con anno di corso più basso

```
SELECT *
FROM S
WHERE ACorso <= ALL (SELECT ACorso
                     FROM S)
```

Subquery correlate: una subquery viene detta correlata se la sua condizione è formulata usando relazioni e/o sinonimi definiti nella query esterna.

Esempio: nome degli studenti che hanno sostenuto l'esame del corso C1

```
SELECT SNome
FROM S
WHERE 'C1' IN (SELECT CC
               FROM E
               WHERE E.Matr = S.Matr)
```

per una maggiore leggibilità è conveniente fare uso di sinonimi.

```
SELECT S1.SNome
FROM S S1
WHERE 'C1' IN (SELECT CC
               FROM E E1
               WHERE E1.Matr = S1.Matr)
```

I sinonimi sono indispensabili quando una stessa relazione compare sia nelle query e esterna che nella subquery (analogamente alla self join).

Per gli attributi non quantificati avviene la quantificazione automatica con il nome della relazione più vicina, ad esempio la precedente interrogazione si può scrivere come:

```
SELECT Città, SNome
FROM S S1
WHERE S1.ACorso >= ALL (SELECT A.Corso
                        FROM S
                        WHERE S1.Città=Città)
```

Il quantificatore esistenziale

Il predicato EXISTS (<subquery>) ha valore true sé e solo se le insieme di valori restituiti da (<subquery>) è non vuoto.

Esempio: nome degli studenti che hanno sostenuto l'esame del corso C1

```
SELECT SNome
FROM S
WHERE EXISTS (SELECT *
              FROM E
              WHERE E.Matr = S.Matr
              AND E.CC = 'C1')
```

Il predicato NOT EXISTS (<subquery>) ha valore true sé e solo se le insieme dei valori restituiti da (<subquery>) è vuoto.

Esempio: nome degli studenti che hanno sostenuto l'esame del corso C1

```
SELECT SNome
FROM S
WHERE NOT EXISTS (SELECT *
                  FROM E
                  WHERE E.Matr = S.Matr
                  AND E.CC = 'C1')
```

Generalmente, al fine di formulare query significative con EXISTS è indispensabile utilizzare subquery correlate.

Riduzioni di query innestate

Le query innestate formulate con i seguenti operatori si possono ridurre a query semplici equivalenti (stessa risposta per ogni possibile istanza delle basi di dati):

- IN
- ANY (con qualsiasi operatore di confronto)
- EXISTS (con subquery correlata)

Le query innestate formulate con i seguenti operatori non si possono ridurre:

- NOT IN
- NOT ANY (con qualsiasi operatore di confronto)
- NOT EXISTS (con subquery correlata)

Funzioni aggregate

SQL mette a disposizione una serie di funzioni per elaborare i valori di un attributo (MAX, MIN, AVG, SUM) e per contare le tutte le che soddisfano una condizione (COUNT).

Opzioni: *: conteggio del numero di righe (COUNT)
ALL: trascura i null (COUNT, AVG, SUM); (default)
DISTINCT: trascura i null ed elimina i duplicati (COUNT, AVG, SUM)

Non è l'è città la presenza contemporanea nella <lista-select> di nomi di campi e funzioni aggregate, pertanto la seguente interrogazione non è corretta:

```
SELECT Matr, MAX (Voto)
FROM E
```

Raggruppamento: la clausola GROUP BY

In una distruzione SELECT è possibile formare dei gruppi di tuple che hanno lo stesso valore di specificati attributi, tramite la clausola GROUP BY:

```
SELECT [DISTINCT | ALL] <lista-select>
FROM <lista-from>
[WHERE <condizione>]
[GROUP BY <lista-group>]
```

Il risultato della SELECT è un **unico record per ciascun gruppo**: pertanto nella <lista-select> possono comparire solo:

- uno o più attributi di raggruppamento, cioè specificati in <lista-group>
- funzioni aggregate: tali funzioni vengono valutate, e quindi forniscono un valore unico, per ciascun gruppo

Raggruppamento: la clausola HAVING

La clausola HAVING è equivalente alla clausola WHERE applicata a gruppi di tuple: ogni gruppo costruito tramite GROUP BY fa parte del risultato dell'interrogazione solo se il predicato specificato nella clausola HAVING risulta soddisfatto.

[GROUP BY <lista-group>]
[HAVING <condizione>]

Il predicato espresso nella clausola HAVING è formulato utilizzando

- uno o più attributi specificati in <lista-group>
- funzioni aggregate

Unione, differenza, intersezione

Unione: <subquery> UNION [ALL] <subquery>
Differenza: <subquery> EXCEPT [ALL] <subquery>
Intersezione: <subquery> INTERSECTION [ALL] <subquery>
Con ALL si considera la semantica del multiset

- (a, b) UNION ALL (a) = (a, a, b)
- (a, a) EXCEPT ALL (a) = (a)
- (a, a) INTERSECTION ALL (a, b) = (a, a)

Le interrogazioni formulate tramite gli operatori EXCEPT e INTERSECTION possono essere prescritte, in maniera equivalente, utilizzando operatori introdottoti in precedenza, (come ad esempio NOT IN e IN rispettivamente).
Questo non è valido per l'operatore UNION. Anche per questo motivo generalmente nelle implementazioni di SQL l'unico degli operatori presenti è UNION.

Divisione

L'operazione di divisione non è definita il SQL. Pertanto, le interrogazioni che richiedono tale operatore, vengono in genere riformulate con una doppia negazione nel seguente modo:

```
SELECT *
FROM S
WHERE NOT EXIST
        (SELECT *
         FROM C
         WHERE CD = 'D1'
         AND NOT EXIST
                (SELECT *
                 FROM E
                 WHERE E.Matr = S.Matr
                 AND E.CC = C.CC))
```

Manipolazione dei dati

Inserimento di record:

- inserimento esplicito di un singolo record:
  INSERT INTO <tabella> [<lista-attr>] VALUES <lista-valori>
- inserimento del risultato di una interrogazione:
  INSERT INTO <tabella> [<lista-attr>] VALUES <subquery>
  dove i valori riportati in <lista-valori> devono corrispondere in numero e tipo agli attributi specificati in <lista-attr> e <lista-attr> deve essere contenuta nella lista di attributi di <tabella>: i valori corrispondenti a gli attributi mancanti vengono posti uguali al valore di default oppure a null.

Modifica di record:

- UPDATE <tabella> SET <attr> = <espressione>
  [WHERE <condizione>]

Cancellazione di record:

- DELETE FROM <tabella>
  [WHERE <condizione>]

Viste

Una vista è una tabella " virtuale" definita, tramite un'interrogazione, da altre tabelle base o da altre viste. Le interrogazioni non vengono eseguite all'atto di creazione della vista, ma quando la vista viene utilizzata. Esse vengono calcolate ogni volta che vengono chiamate, quindi sono sempre aggiornate.
Le viste aggiornabili, sono possibili operazioni di modifica (INSERT, UPDATE, DELETE), tradotte in operazioni di modifica sulle tabelle base.
In SQL una vista è aggiornabile solo quando una sola riga di ciascuna tabella di base corrisponde ad una riga della vista.
In particolare, quindi **non** solo aggiornabili viste ottenute:

- Tramite GROUP BY e funzioni aggregate;
- Tramite DISTINCT senza inclusione di una chiave;
- Tramite riferimenti a viste non aggiornabili;