SQL

- Structured Query Language

Si basa su un modello RELAZIONALE, in cui le relazioni sono rappresentate da TABELLE.
In generale una tabella è un multiinsieme di tuple (ci possono essere duplicati), ma possono essere espressi vincoli di chiave (primary key o unique)

Ogni tabella si denota con NomeSchema.NomeTabella.
Quando l'ambiguità sullo schema non sussiste si usa direttamente NomeTabella

Ogni attributo di una tabella si denota con
NomeSchema.NomeTabella.NomeAttributo
Valgono le stesse regole di ambiguità sulla tabella

SINTASSI SELECT

select SQL ::=

```
select   <targetList>
from     ListaTabelle
[ where     condizioni semplici ]
[ group by  ListaAttributiDiRaggruppamento ]
[ having    condizioniAggregati ]
[ order by  ListaAttributiDiOrdinamento ]
[ limit     numero ]
```

SELECT

```
select   attributo, ..., attributo
from     tabella1, ..., tabella n
where    condizione
```

Semantica:
le operazioni che effettua il DBMS sono equivalenti a: seguenti passaggi:

1. Si effettua il prodotto cartesiano delle tabelle nella clausola from
2. Si verifica, per ogni tupla, la condizione nella clausola where, effettuando una SELECTION del linguaggio relazionale
3. Si fanno la proiezione sugli attributi nella select (target list)

equivalente a $PROJ_{attr_1,\dots,attr_i}(SEL_{condizione}(t_1 \times \dots \times t_n))$

Alcuni modificatori ammessi

- usare l'asterisco "*" nella select vuol dire non effettuare alcuna proiezione
- si può inserire "distinct" così che, a discapito delle chavi, le tuple sugli attributi selezionati saranno univoche
- nella target list è possibile inserire espressioni
- la condizione può essere concatenata ad altre attraverso IN, OR, AND...

JOIN NELLA FROM

è possibile effettuare un join nella clausola from in condizione (come equijoin)

```
select ...
from Tabella1 join Tabella2 on condizionejoin
[where altracondizione]
```

Si possono effettuare altri tipi di join

- left outer join: fai il join sugli attributi della tabella di sinistra. Se l'attributo collima con la tupla di destra allora mempi i dati, altrimenti inserisci NULL
- right outer join: simile alla left, ma con tabella al contrario
- full outer join: tutte le tuple appariranno, collimanti o meno.

ORDER BY

C'è la possibilità di ordinare le tuple secondo uno o più attributi. Si mette sotto la where

OPERATORI AGGREGATI

Nelle espressioni della target list si possono avere anche espressioni che calcolano valori a partire da insiemi di tuple come **conteggio, minimo, massimo, media, somma totale**

sintassi

Funzione ([Distinct] EsprsuAttributi)

count

conta il numero di tuple (count(*)) oppure i valori di un attributo, considerando o meno i duplicati (count ([distinct] Attributo)

sum, avg, max, min

Ammettono come valori espressione o un attributo, ma non *. sum e avg devono essere di tipo numerico o di tempo, mentre per max e min ci deviessere un ordinamento

RAGGRUPPAMENTI

In molti casi vorremmo che gli operatori aggregati venissero applicati a gruppi di tuple, e per specificarli si utilizza la clausola "group by"

```
group by listaAttributi
```

Semantica

```
select <target list>
from R
group by Ai
```

1. Si esegue l'interrogazione ignorando la group by e la target list

2. Sulle tuple che risultano si formano i gruppi, raggruppando per tuple che hanno lo stesso valore per A_i.
   Si produce nel risultato una tupla per ogni gruppo, e per ognuna di esse si applica la target list, assieme ad eventuali operatori aggregati

N.B. in un'interrogazione che fa uso di group by dovrebbero comparire target list omogenee, ovvero quando attributi che compaiono nella group by

es.

```
select eta, avg(reddito)
from persone
group by eta
```

Having

Si possono imporre delle condizioni di selezione sui gruppi, la selezione è diversa e appunto si usa having, che appare dopo la group by.
Su di essa si inseriscono condizioni anche sugli operatori di aggregazioni

UNIONE, INTERSEZIONE, DIFFERENZA

La select da sola non permette l'unione, quindi serve un costrutto specifico

union

```
select ...
union [all]
select ...
```

N.B. le target list delle due select devono avere lo stesso numero d. elementi, e union da solo eliminerà gli attributi

except

effettua la differenza

```
select ...
except
select...
```

N.B. le target list delle due select devono avere lo stesso numero d. elementi.

intersect

```
select ...
intersect
select...
```

N.B. le target list delle due select devono avere lo stesso numero d. elementi.

DEFINIZIONE DEI DATI

create table

istruzione più importante, serve a definire nuove tabelle e fa 2 cose
- definisce uno schema di relazione
- crea un istanza vuota di tabella

Sintassi:

```
create table NomeTabella (
        NomeAttributo Dominio [vincoli],
        :
        NomeAttributo Dominio [vincoli],
        [altri vincoli]
)
```

Domini predefiniti:

- Caratteri:
  - char(n) - stringhe lunghezza fissa
  - varchar(n) - stringa lunghezza variabile, max n
  - nchar(n) e nvarchar(n) - come sopra ma UNICODE
- Numerici:
  - int, smallint - interi
  - numeric - valori numerici esatti non negativi
  - decimal - valori numerici anche negativi
  - float, real, double precision - reali
- Data, ora, intervalli di tempo:
  - date, time, time stamp
  - time with timezone, timestamp with timezone
- Bit
  - bit(n)
  - bit varying(n)

VINCOLI IN SQL

Si vedranno diversi tipi di vincoli, sia intrarelazionali che interrelazionali.
Per definire un vincolo bisogna usare constraint <nomeVincolo>, non è obbligatorio inserire un nome, ma fortemente consigliato poichè se ci sarà un errore su vincoli non soddisfatti il DBMS darà il nome del vincolo.

Vincoli intrarelazionali

- **Not null** su singoli attributi

- **unique** permette di definire un insieme di attributi come superchiave
  - singolo attributo: dopo specifica del dominio
  - più attributi: unique (Attributo, ..., Attributo)

- **primary key**: definizione chiave primaria, al massimo una per ogni tabella per uno o più attributi. Definite come la unique

- **check**: per vincoli di tupla o interrelazionali

Vincoli interrelazionali

- **check** per vincoli complessi

- **references** e **foreign key** permettono di definire vincoli di integrità referenziali
  - per singoli attributi: references dopo il dominio
  - su più attributi: foreign key (attributo, ..attributo) references...

  gli attributi referenziati, nella tabella di arrivo devono formare una chiave (primary key o unique). Se mancano, il riferimento si intende alla chiave primaria

Alter table

permette di modificare una tabella, effettuando le seguenti operazioni:

- change o modify
- drop
- add

la alter table è importante per definire tabelle in cui sono definiti vincoli di integrità referenziali "incrociati", ovvero che le tabelle sono "fortemente referenziati".

in tal caso si può procedere nel seguente modo:
- si definisce R_1 senza vincoli di integrità referenziali verso R_2
- si fornisce la definizione di R_2 completa
- con alter table si aggiunge alla definizione di R_1 i vincoli di integrità referenziali verso R_2

Drop table

elimina una tabella

drop table nometabella restrict | cascade

con restrict si elimina la tabella solo se non ci sono riferimenti ad essa

con cascade elimina la tabella e tutte le tabelle che si riferiscono ad essa

MANIPOLAZIONE DATI

SI possono effettuare operazioni di:

- inserimento — insert
- eliminazione — delete
- modifica — update

insert

```
insert into Tabella [(Attributi)]
values (valori)

oppure

insert into Tabella [(attributi)]
select ...
```

delete

```
delete from Tabella [where condizione]
```

update

```
update NomeTabella
set   Attributo = <espressione | select... | null | default>
[where condizione]
```

INTERROGAZIONI COMPLESSE

In generale, nelle condizioni atomiche (spesso nella where) può comparire una select tra parentesi.
Si effettuerà quindi un'operazione ANNIDATA

Le condizioni permettono quindi:
- confronto tra un attributo e il risultato di una sottointerrogazione
- quantificatori esistenziali

operatori:

Il risultato di un'interrogazione può essere oggetto di clausole di confronto nella where mediante diversi operatori. Non è detto che il risultato della select annidata sia una sola tupla. La comparazione diretta tra una tupla ed un insieme non è possibile.
Si usa quindi

- in, che è equivalente ad = any
- not in che è equivalente a <> all
- exists per verificare che la tabella risultante dalla select non sia vuota

Interrogazioni annidate nella clausola from, o VISTE INLINE

Una vista è una tabella le cui tuple sono derivate da altre tabelle mediante un'interrogazione.

è possibile definirle in due modi:

```
select <target list>
from Tabella, ..., Tabella, (select ...
                               from ...
                               where ...) nomeVista
```

oppure anche come

```
with nomeVista as
   (select ...
     from ...
     where ...)
select ..
from
where
```

} si può usare nomeVista come tabella

ALTRE OPERAZIONI

creazione viste

possiamo definire una vista nella base di dati:

```
create view NomeVista[(listaAttributi)] as SelectSQL
```

è una tabella virtuale, ergo viene effettivamente creata solo una volta che è invocata

TRANSAZIONI

Sono un insieme di operazioni considerate come indivisibili; "atomiche", anche in presenza della concorrenza.

godono della proprietà ACID

- Atomicità
- Consistenza
- Isolamento
- Durabilità (resistenza)

Ogni istruzione SQL non all'interno di una transazione definita esplicitamente è una transazione.
Per definire esplicitamente si usa

- begin (o begin transaction)
- commit (o commit work, end o end transaction)
- rollback (o rollback work)

È comodo quando si sono 2 tabelle fortemente referenziate e si vuole aggiungere una tupla ad entrambi "contemporaneamente"