

# Il linguaggio SQL (Structured Query Language). Fondamenti e Pattern d'uso

# Riferimenti

- Per una presentazione sistematica dei comandi di SQL, potete fare riferimento ai seguenti siti:
  - Manuale introduttivo (interattivo) a MySQL:  
<https://www.w3schools.com/mysql/default.asp>
  - *Reference manual* ufficiale di MySQL:  
<https://dev.mysql.com/doc/refman/8.4/en/tutorial.html>
  - Guida all'uso di MySQL:  
<https://www.javatpoint.com/mysql-tutorial>
- Nelle lezioni discuteremo soprattutto di pattern complessi e casi particolari di uso di query SQL

# Strumenti per lavorare con RDBMS

- Installare sul proprio computer un DBMS a scelta tra:
  - MariaDB: <https://mariadb.org/download/>
  - MySQL: <https://www.mysql.com/downloads/>
  - [PostgreSQL: <https://www.postgresql.org/download/> ]
- Installare un'interfaccia grafica per l'utilizzo del DBMS:
  - MySQL workbench: <https://dev.mysql.com/downloads/workbench/>
  - Dbeaver: <https://dbeaver.io/download/> («a cross-platform database tool for developers, database administrators, analysts, and everyone working with data. It supports all popular SQL databases like MySQL, MariaDB, PostgreSQL, SQLite, Apache Family, and more»)
  - [pgAdmin: <https://www.pgadmin.org/> ]

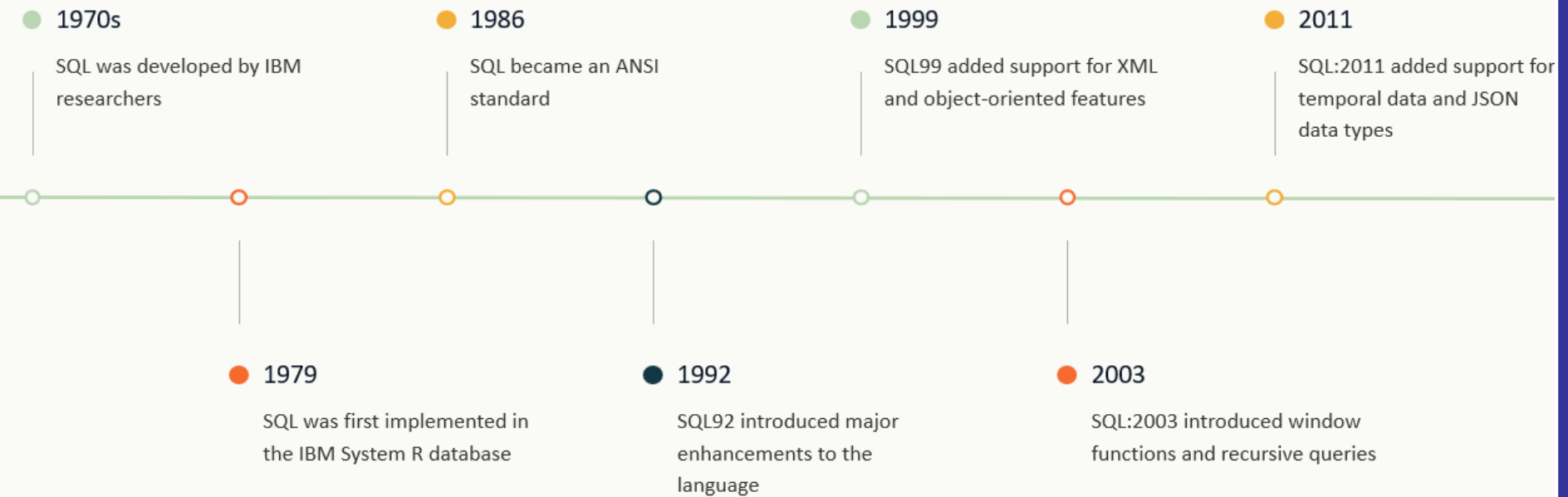
# Schemi e dati per esercitarsi

- Sul Classroom del corso sono a disposizione gli script SQL per generare schema e dati di due database relazionali:
  - La base di dati «**Sailors**», che è lo schema utilizzato nel libro di testo e nelle slide per presentare SQL
  - La base di dati «**NorthWind**», che è uno schema pubblico con una buona quantità di dati nel settore degli ordini di un'azienda
- Per entrambi gli schemi verranno fornite liste di esercizi (query)

# SQL: definizione

- SQL (*Structured Query Language*) è un linguaggio specializzato per la gestione di basi di dati relazionali
- È utilizzato per la creazione, la modifica e l'interrogazione dei dati
- Standardizzato e ampiamente utilizzato nell'ambito della gestione dei dati

# SQL: breve storia



# SQL: DDL e DML

- DDL (Data Definition Language): Utilizzato per definire la struttura della base di dati
  - Creazione di tabelle, modifiche dello schema, definizione di vincoli.
- DML (Data Manipulation Language): Utilizzato per manipolare i dati all'interno del database.
  - Inserimento, aggiornamento, cancellazione e interrogazione dei dati.

# Istanza di DB utilizzata in queste slide

<i>sid</i>	<i>sname</i>	<i>rating</i>	<i>age</i>
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Figure 5.1 An Instance *S3* of Sailors

<i>sid</i>	<i>bid</i>	<i>day</i>
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Figure 5.2 An Instance *R2* of Reserves

<i>bid</i>	<i>bname</i>	<i>color</i>
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Figure 5.3 An Instance *B1* of Boats



# **STRUTTURA GENERALE DELLE QUERY SQL**

# Struttura base

SELECT	[DISTINCT] <i>target-list</i>
FROM	<i>relation-list</i>
WHERE	<i>qualification</i>

- *relation-list*: lista di nomi di relazioni (talvolta con le cosiddette *variabili di range*) – da 1 a  $n$
- *target-list*: lista di attributi che appartengono alle relazioni nella *relation-list*
- *qualification*: predicati di confronto (  $<, >, =, \leq, \geq, \neq$  ) eventualmente combinati con AND, OR o NOT
- DISTINCT: chiave opzionale per indicare che i duplicati devono esser eliminati dal risultato dell'interrogazione (il *default* è che NON vengono eliminati!!)

# Strategia concettuale di valutazione

SELECT	[DISTINCT] <i>target-list</i>
FROM	<i>relation-list</i>
WHERE	<i>qualification</i>

- Si calcola il prodotto cartesiano delle tabelle nella *relation-list*
- Si scartano le tuple che non soddisfano le *qualifications*
- Si cancellano le colonne degli attributi che non appartengono alla *target-list*
- Se è presente la chiave DISTINCT, si eliminano i duplicati

NB: questa è una strategia **molto inefficiente** per valutare la query!! Il modulo di *query optimization* dei DBMS serve proprio a creare piani di esecuzione più efficienti

# Esempi

```
select s.sname , s.rating  
from sailors s  
where rating > 7;
```



	ABC sname ▼	123 rating ▼
1	lubber	8
2	Andy	8
3	Rusty	10
4	Zorba	10
5	Horatio	9

```
select s.sname, r.bid  
from sailors s, reserves r  
where s.sid = r.sid ;
```



	ABC sname ▼	123 bid ▼
1	Dustin	101 ↗
2	Horatio	101 ↗
3	Dustin	102 ↗
4	lubber	102 ↗
5	Horatio	102 ↗
6	Dustin	103 ↗
7	lubber	103 ↗
8	Horatio	103 ↗
9	Dustin	104 ↗
10	lubber	104 ↗

# Esempi

```
select s.sid
from sailors s, reserves r
where s.sid = r.sid and r.bid = 103;
```



	<u>123 sid</u> ▼
1	22
2	31
3	74

```
select s.sid , b.bname , b.color
from sailors s, reserves r , boats b
where s.sid = r.sid and r.bid = b.bid ;
```




	<u>123 sid</u> ▼	<u>ABC bname</u> ▼	<u>ABC color</u> ▼
1	22	Interlake	blue
2	64	Interlake	blue
3	22	Interlake	red
4	31	Interlake	red
5	64	Interlake	red
6	22	Clipper	green
7	31	Clipper	green
8	74	Clipper	green
9	22	Marine	red
10	31	Marine	red

# Operatori di confronto

- Quando viene creata una qualificazione, questa si basa quasi sempre sull'uso di operatori di confronto
- In SQL i principali sono:
  - Operatori aritmetici: =, <, >, >=, <=, <>
  - **BETWEEN**: verifica se un valore è compreso in un intervallo
  - **IN**: verifica se un valore è compreso in una lista
  - **LIKE**: confronto fra stringhe con caratteri jolly (% e \_)
  - **IS [NOT] NULL**: verifica se un valore ha (o non ha) valore NULL □ [... **where x = NULL**] è **SBAGLIATO!!**


# Esempi

```
select * from reserves r
where r.`day` between '1998-06-02' and '1998-08-31'
order by r.`day` ;
```




sid	bid	day
31	103	1998-06-11
22	104	1998-07-10
64	102	1998-08-09
74	103	1998-08-09
22	103	1998-08-10

```
select r.sid , r.`day`
from reserves r
where r.bid in(101, 103) ;
```




sid	day
22	1998-10-10
64	1998-05-09
22	1998-08-10
31	1998-06-11
74	1998-08-09

```
select s.sname
from sailors s
where s.sname like '%s_%' ;
```



sname
Dustin
Rusty

```
select distinct s.sid, r.bid
from sailors s left join reserves r
on s.sid = r.sid
where r.bid is null;
```



sid	bid
29	[NULL]
32	[NULL]
58	[NULL]
71	[NULL]
85	[NULL]
95	[NULL]

# Operatori di aggregazione

- SQL supporta 5 operatori di aggregazione:
  - `count(*)`: numero di tuple restituite da una query
  - `count([DISTINCT] A)`: numero di valori (unici) di A
  - `sum([DISTINCT] A)`: somma dei valori (unici) di A
  - `avg([DISTINCT] A)`: media dei valori (unici) di A
  - `MAX(A)`: il valore più alto nella colonna A
  - `MIN(A)`: il valore più basso nella colonna A
- Questi operatori permettono di aggregare dati usando funzioni aritmetiche, andando oltre gli operatori dell'algebra relazionale



# Esempi

```
select min(s.rating) as ValMinima,  
       max(s.rating) as ValMassima,  
       avg(s.rating) as ValMedia  
from sailors s ;
```



	123 ValMinima	123 ValMassima	123 ValMedia
1	1	10	6,6

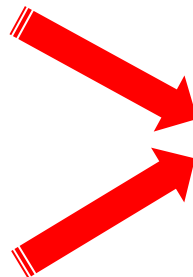
```
select count(*) as NumNoleggi  
from reserves r  
where r.bid = 103;
```



	123 NumNoleggi
1	3

```
select count(s.sname)  
from sailors s ;
```

```
select count(distinct s.sname)  
from sailors s ;
```



Che differenza c'è?

# Struttura estesa: GROUP BY / HAVING

- Talvolta è necessario applicare operazioni di aggregazione non a tutte le tuple di una relazione, ma a gruppi di tuple raggruppate in base a un certo criterio
  - Esempio 1: trovare quante volte è stata noleggiata ogni singola barca disponibile
  - Esempio 2: trovare l'età media dei marinai che hanno lo stesso rating nella tabella Sailors
- A questo scopo, la struttura di base delle query è estesa con le clausole GROUP BY / HAVING

# GROUP BY e HAVING

```
SELECT    [ DISTINCT ] select-list
FROM      from-list
WHERE     qualification
GROUP BY  grouping-list
HAVING    group-qualification
```

- **select-list**: è composta da (1) una lista di nomi di colonne e (2) una lista di termini della forma **agg-op**(*nome-colonna*) *AS nuovo\_nome* (il nuovo nome è utile, anche se non indispensabile)
- Ogni colonna che compare in (1) **deve** apparire anche nella **grouping-list** (onde evitare che nel gruppo quella colonna possa avere valori diversi)
- Le espressioni che compaiono nella **group-qualification** devono sempre avere un solo valore per gruppo (valore che viene usato per decidere se un gruppo deve essere restituito nel risultato)
- Se nella query non compare il GROUP BY, l'intera tabella viene considerata come un singolo gruppo

# GROUP BY e HAVING: valutazione

```
SELECT    [ DISTINCT ] select-list  
FROM      from-list  
WHERE     qualification  
GROUP BY  grouping-list  
HAVING    group-qualification
```

1. Costruzione del prodotto cartesiano indicate nel FROM
2. Applicazione delle qualificazioni (WHERE)
3. Eliminazione delle colonne non desiderate (che non compaiono nella **select-list**, nella **grouping list** e nelle **group qualifications**)
4. Riordino della relazione in base al GROUP BY, in modo da identificare i gruppi
5. Applicazione delle **group qualifications** presenti nella clausola HAVING
6. Generazione le righe di risposta per ogni gruppo rimanente

# Esempi GROUP BY / HAVING

Per ogni rating, trovare l'età media dei marinai con quel rating

```
select s.rating, avg(s.age) as EtaMedia  
from sailors s;
```

**Errore!!**



	123 rating ▼	123 EtaMedia ▼
1	7	36,6

```
select s.rating, avg(s.age) as EtaMedia  
from sailors s  
group by s.rating ;
```



	123 rating ▼	123 EtaMedia ▼
1	1	33
2	3	43
3	7	40
4	8	40,5
5	9	35
6	10	25,5

```
select r.bid, count(*) as NumNoleggi  
from reserves r  
group by r.bid  
having NumNoleggi > 2;
```



	123 bid ▼	123 NumNoleggi ▼
1	102 ↗	3
2	103 ↗	3

# GROUP BY e HAVING: esempio

(Q32) Find the age of the youngest sailor who is eligible to vote (i.e., is at least 18 years old) for each rating level with at least two such sailors.

```
SELECT  S.rating, MIN (S.age) AS minage
FROM    Sailors S
WHERE   S.age >= 18
GROUP BY S.rating
HAVING  COUNT (*) > 1
```

sid	sname	rating	age
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5
96	Frodo	3	25.5

Passi 1, 2 e 3



rating	age
7	45.0
1	33.0
8	55.5
8	25.5
10	35.0
7	35.0
9	35.0
3	25.5
3	63.5
3	25.5

Passo 4



rating	age
1	33.0
3	25.5
3	25.5
3	63.5
7	45.0
7	35.0
8	55.5
8	25.5
9	35.0
10	35.0

Passi 5 e 6



rating	minage
3	25.5
7	35.0
8	25.5

# **ANNIDAMENTO DI QUERY**

# Interrogazioni annidate (*nested queries*)

- Una query annidata è una query (detta *interna*) che compare in un'altra query (detta *esterna*)
- Per esempio:

```
select s.sname
from sailors s
where s.rating = (select max(s2.rating)
                  from sailors s2 );
```

- Semanticamente, l'idea è simile a quella di un *nested loop*: per ogni tupla di Sailor nella *query esterna*, viene valutata la condizione *where* eseguendo la *query interna*



# Interrogazioni annidate (*nested queries*)

- Una query annidata può restituire:
  - Un singolo valore (con cui confrontare un valore di un attributo della query esterna)

```
select s.sname
from sailors s
where s.rating = (select max(s2.rating)
                  from sailors s2 );
```

- Un insieme di valori tra i quali vogliamo verificare se vale una certa condizione (normalmente espressa con gli operatori IN, EXISTS, ANY o ALL)

```
select s.sname , s.rating
from sailors s
where s.sid in (select distinct r2.sid
               from reserves r2);
```

# Query indipendenti e correlate

- Una query annidata si dice:
  - **Indipendente:** quando può essere eseguita in modo autonomo, senza fare riferimento a nessuna colonna o valore della query esterna.
  - **Correlata:** quando la query interna fa riferimento a una o più colonne della query esterna. Questo significa che la subquery deve essere rieseguita per ogni riga elaborata dalla query esterna
- Differenze principali:
  - **Esecuzione:** una subquery indipendente viene eseguita una sola volta, e il suo risultato viene utilizzato dalla query esterna, mentre una subquery correlata deve essere rieseguita per ogni riga elaborata dalla query esterna
  - **Indipendenza:** le subquery indipendenti possono essere eseguite anche da sole, mentre le query correlate non hanno senso al di fuori della query esterna.

# Esempi

- **Subquery indipendente:** trovare i nomi dei marinai che hanno un rating minore rispetto al rating medio

```
select s.sname
from sailors s
where s.rating < (select avg(s2.rating)
                  from sailors s2 );
```

- **Subquery correlata:** trovare i nomi dei marinai che hanno prenotato la barca #103

```
select s.sname
from sailors s
where exists (select * from sailors s2, reserves r
              where r.bid = 103 and
              s.sid = r.sid);
```

# Casi d'uso

- Usare in un filtro valori calcolati dinamicamente
  - Esempio: marinai che hanno un rating minore rispetto al rating medio di tutti i marinai

```
select s.sname
from sailors s
where s.rating < (select avg(s2.rating)
                  from sailors s2 );
```



	ABC sname ▼
1	Brutus
2	Art
3	Bob

- Perché non si poteva fare semplicemente così?

```
select s.sname
from sailors s
where s.rating = avg(s2.rating);
```



# Casi d'uso

- Eseguire confronti con risultati derivati
  - Esempio: marinaio / marinai che ha / hanno il rating più alto

```
select s.sname
from sailors s
where s.rating = (select max(s2.rating)
                  from sailors s2 );
```



	ABC sname ▼
1	Rusty
2	Zorba

- Applicare filtri che dipendano dalla verifica che certi valori esistano (o non esistano)
  - Esempio: trovare i marinai che non hanno mai fatto neanche una prenotazione

```
select s.sid
from sailors s
where not exists (select r.sid from reserves r
                  where r.sid = s.sid);
```



	123 sid ▼
1	29
2	32
3	58
4	71
5	85
6	95

# Casi d'uso

- Eseguire operazioni di aggregazione condizionale
  - Esempio: rating dei marinai che hanno effettuato meno di 2 prenotazioni

```
select s.sid , s.rating
from sailors s
where 2 > (select count(*)
           from reserves r2
           where r2.sid = s.sid);
```



	sid	rating
1	29	1
2	32	8
3	58	10
4	71	10
5	74	9
6	85	3
7	95	3

- Unire i risultati di una subquery con altre tabelle usando una JOIN
  - Esempi nel seguito

# Interrogazioni annidate a più livelli

- È possibile avere più di un livello di annidamento
  - Esempio: marinai che hanno / non hanno prenotato una barca rossa:

```
SELECT S.sname
FROM   Sailors S
WHERE  S.sid IN ( SELECT R.sid
                  FROM   Reserves R
                  WHERE  R.bid IN ( SELECT B.bid
                                  FROM   Boats B
                                  WHERE  B.color = 'red' ) )
```

[illegible]

# Interrogazioni annidate e operatori insiemistici di confronto

- Esempio: trovare i marinai che hanno un rating più alto di quello di uno qualsiasi tra i marinai che si chiamano Horatio

```
select *  
from sailors s  
where s.rating > any (select s2.rating  
                      from sailors s2  
                      where s2.sname = 'Horatio');
```



	123 sid	abc sname	123 rating	123 age
1	31	lubber	8	55,5
2	32	Andy	8	25,5
3	58	Rusty	10	35
4	71	Zorba	10	16
5	74	Horatio	9	35

- Basta che un marinaio si chiami Horatio per rendere la clausola ANY vera (perché compare Horatio nei risultati?)
- Se non ce ne sono, ANY restituisce FALSE e viene restituito l'insieme vuoto di tuple come risposta alla query



# Interrogazioni annidate e operatori insiemistici di confronto

- Esempio 2: trovare i marinai che hanno un rating più alto di quello di tutti i marinai che si chiamano Horatio:

```
select *  
from sailors s  
where s.rating < all (select s2.rating  
                     from sailors s2  
                     where s2.sname = 'Horatio');
```



	sid	sname	rating	age
1	58	Rusty	10	35
2	71	Zorba	10	16

- Qui la condizione deve valere per tutti i marinai chiamati Horatio
- NB: se non ce ne sono, ALL restituisce TRUE e viene restituito l'insieme di tutte le tuple di Sailor

# Operatori insiemistici

- Servono per effettuare operazioni insiemistiche tra due o più insiemi (compatibili) di tuple generate da altrettante query
- Essi sono:
  - **UNION (UNION ALL)**: fa l'unione di due insiemi di tuple. UNION elimina gli eventuali duplicati, UNION ALL no
  - **INTERSECT**: restituisce solo le tuple comuni a due insiemi di tuple
  - **EXCEPT**: restituisce le tuple del primo insieme a patto che non compaiono nel secondo insieme (*set difference*)

# Esempi di query con op. insiemistici

- Stabilire cosa restituiscono le seguenti query:

```
select s.sid from sailors s  
except  
select r.sid from reserves r ;
```



--

```
select s.sid from sailors s  
intersect  
select r.sid from reserves r ;
```



--

```
select r.sid  
from reserves r JOIN boats b on r.bid = b.bid  
where b.color = 'red'  
union  
select r.sid  
from reserves r JOIN boats b on r.bid = b.bid  
where b.color = 'blue';
```



--

# Dove possono essere annidate le query

- Le *sub query* possono essere annidati in diversi punti di un'interrogazione SQL:
  - Nella clausola **FROM**: serve a costruire una tabella temporanea su cui eseguire il resto della query
  - Nella clausola **WHERE**: serve a calcolare al volo un singolo valore su cui operare un confronto o un insieme di valori da confrontare con un valore già disponibile
  - Nella clausola **HAVING** (vedi prossime slide): serve a calcolare un valore per filtrare i gruppi che devono essere restituiti (simile al WHERE, ma si opera su gruppi e non su righe)

# JOIN(s)

## ■ Consideriamo le 4 principali forme di JOIN:

1. (INNER) JOIN: la condizione del JOIN è soddisfatta dalle tuple di entrambe le tabelle
2. LEFT (OUTER) JOIN: restituisce tutte le tuple della tabella di sinistra e solo quelle che soddisfano la condizione del JOIN della relazione di destra
3. LEFT (OUTER) JOIN: restituisce tutte le tuple della tabella di sinistra e solo quelle che soddisfano la condizione del JOIN della relazione di destra
4. FULL (OUTER) JOIN\*: tutte le tuple ogni volta che c'è un match in almeno una delle due tabelle



\* MariaDB (come molti altri DBMS) non supporta il FULL OUTER JOIN!

# INNER JOIN vs. PRODOTTO CARTESIANO

- Quando dobbiamo eseguire un INNER JOIN, abbiamo sempre due approcci alternative:
  1. Usare esplicitamente l'operatore JOIN ... ON:

```
select s.sid , r.bid , r.`day`  
from sailors s join reserves r on s.sid = r.sid  
where r.bid = 103;
```

2. Mettere le tabelle interessate nella *relation-list* e specificare le condizioni utilizzando la clausola WHERE:

```
select s.sid , r.bid , r.`day`  
from sailors s, reserves r  
where s.sid = r.sid and r.bid = 103;
```

# INNER JOIN vs. PRODOTTO CARTESIANO

- Le due query restituiscono esattamente lo stesso risultato:

```
select s.sid , r.bid , r.`day`  
from sailors s join reserves r on s.sid = r.sid  
where r.bid = 103;
```

```
select s.sid , r.bid , r.`day`  
from sailors s, reserves r  
where s.sid = r.sid and r.bid = 103;
```



	<small>123</small> sid ▾	<small>123</small> bid ▾	<small>🕒</small> day ▾
1	22	103 <a href="#">🔗</a>	1998-08-10
2	31	103 <a href="#">🔗</a>	1998-06-11
3	74	103 <a href="#">🔗</a>	1998-08-09

- Anche se in teoria l'uso del comando JOIN ... ON è raccomandato (soprattutto per chiarezza nel distinguere la condizione di JOIN dagli altri filtri), nella pratica è possibile utilizzare le due forme in modo interscambiabile
- Diverso è il caso in cui sia necessario usare forme di OUTER JOIN

# LEFT / RIGHT OUTER JOIN

- Immaginiamo ora di voler conoscere quante prenotazioni ha fatto ogni marinaio
- Se usiamo l'*inner join* o il prodotto cartesiano, ovviamente otteniamo questo risultato:

```
select s.sid , count(*)  
from sailors s inner join reserves r  
on s.sid = r.sid  
group by s.sid ;
```

```
select s.sid , count(*)  
from sailors s , reserves r  
where s.sid = r.sid  
group by s.sid ;
```



	125 sid ▼	123 count(*) ▼
1	22	4
2	31	3
3	64	2
4	74	1



# LEFT / RIGHT OUTER JOIN

- Tuttavia, questo risultato non risponde alla nostra richiesta, perché non ci sono i marinai che non hanno mai prenotato una barca.
- Per farlo, dobbiamo usare un LEFT (OUTER) JOIN:

```
select s.sid , count(r.sid)
from sailors s left join reserves r
on s.sid = r.sid
group by s.sid ;
```



	<small>123</small> sid	<small>123</small> count(r.sid)
1	22	4
2	29	0
3	31	3
4	32	0
5	58	0
6	64	2
7	71	0
8	74	1
9	85	0
10	95	0

# LEFT / RIGHT OUTER JOIN

- Immaginiamo ora di voler conoscere da quanti marinai è stata prenotata ogni barca
- In questo caso, usiamo un RIGHT (OUTER) JOIN:

```
select b.bid, count(r.sid)
from sailors s
join reserves r on s.sid = r.sid
right join boats b on b.bid = r.bid
group by b.bid ;
```

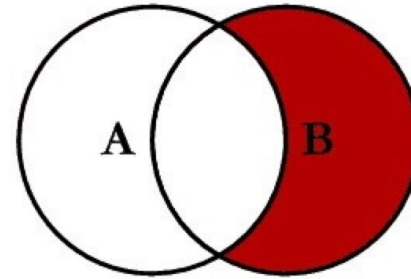
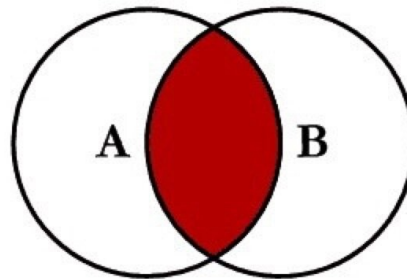
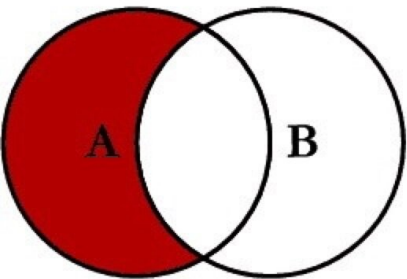
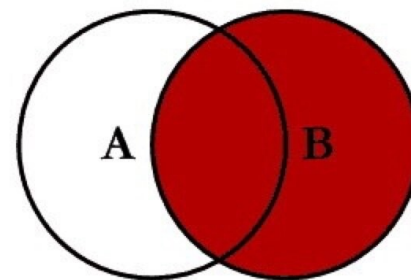
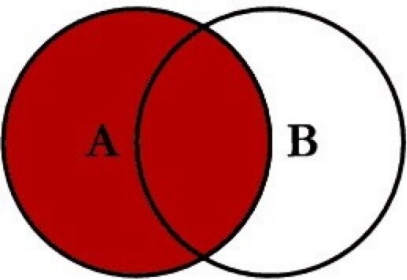


	bid	count(r.sid)
1	101	2
2	102	3
3	103	3
4	104	2

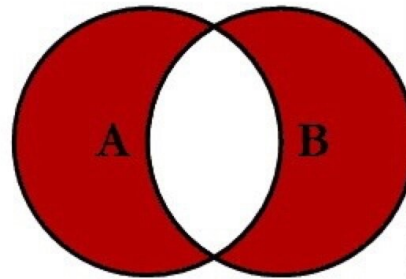
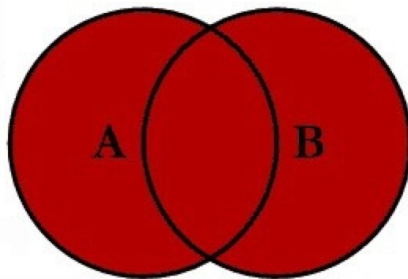
- Con questa specifica istanza della BdD, sarebbe cambiato qualcosa se avessimo usato l'INNER JOIN al posto del RIGHT JOIN? Perché?

# SQL JOIN(s) - II

## SQL JOINS



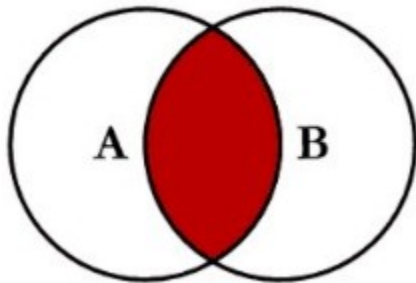
Cosa c'è di diverso rispetto ai casi precedenti?



# SQL JOIN(s)

- I tre nuovi casi si differenziano da quelli visti finora perché NON includono le tuple che hanno una corrispondenza con una tupla dell'altra relazione
- Vengono talvolta chiamati LEFT / RIGHT /FULL ANTI JOIN
- Sono poco usati nello sviluppo di applicazioni, ma sono molto utili in progetti di *business intelligence* o *data science*
- Come possiamo ottenere gli ANTI JOIN dai JOIN che sappiamo usare?

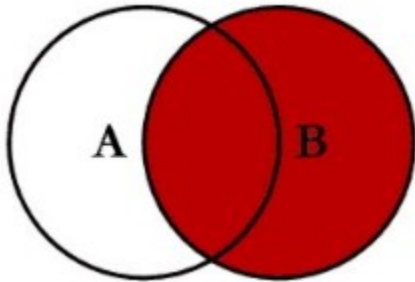
# Tutti i marinai che hanno noleggiato una barca (INNER JOIN)



```
select *  
from reserves r join sailors s  
on r.sid = s.sid;
```

	<small>123</small> sid	<small>123</small> bid	<small>🕒</small> day	<small>123</small> sid	<small>ABC</small> sname	<small>123</small> rating	<small>123</small> age
1	22	101	1998-10-10	22	Dustin	7	45
2	64	101	1998-05-09	64	Horatio	7	35
3	22	102	1998-10-10	22	Dustin	7	45
4	31	102	1998-10-11	31	lubber	8	55,5
5	64	102	1998-08-09	64	Horatio	7	35
6	22	103	1998-08-10	22	Dustin	7	45
7	31	103	1998-06-11	31	lubber	8	55,5
8	74	103	1998-08-09	74	Horatio	9	35
9	22	104	1998-07-10	22	Dustin	7	45
10	31	104	1998-01-11	31	lubber	8	55,5

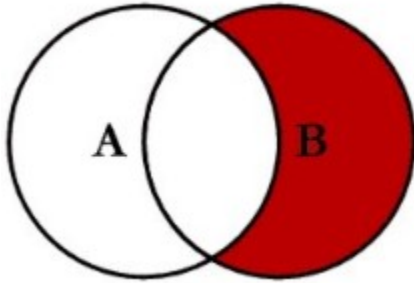
# I noleggi effettuati da tutti i marinai noti (RIGHT JOIN)



```
select *  
from reserves r right join sailors s  
on r.sid = s.sid;
```

	123 sid	123 bid	day	123 sid	abc sname	123 rating	123 age
1	22	101	1998-10-10	22	Dustin	7	45
2	22	102	1998-10-10	22	Dustin	7	45
3	22	103	1998-08-10	22	Dustin	7	45
4	22	104	1998-07-10	22	Dustin	7	45
5	[NULL]	[NULL]	[NULL]	29	Brutus	1	33
6	31	102	1998-10-11	31	lubber	8	55,5
7	31	103	1998-06-11	31	lubber	8	55,5
8	31	104	1998-01-11	31	lubber	8	55,5
9	[NULL]	[NULL]	[NULL]	32	Andy	8	25,5
10	[NULL]	[NULL]	[NULL]	58	Rusty	10	35
11	64	101	1998-05-09	64	Horatio	7	35
12	64	102	1998-08-09	64	Horatio	7	35
13	[NULL]	[NULL]	[NULL]	71	Zorba	10	16
14	74	103	1998-08-09	74	Horatio	9	35
15	[NULL]	[NULL]	[NULL]	85	Art	3	22,5
16	[NULL]	[NULL]	[NULL]	95	Bob	3	63,5

# Tutti i marinai che non hanno noleggiato barche (RIGHT ANTI JOIN)

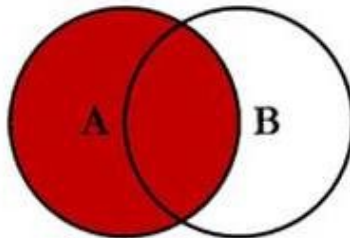


```
select *  
from reserves r right join sailors s  
on r.sid = s.sid  
where r.sid is NULL;
```

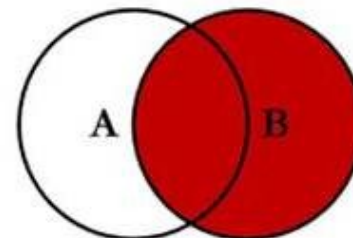
	<small>123</small> sid	<small>123</small> bid	<small>🕒</small> day	<small>123</small> sid	<small>ABC</small> sname	<small>123</small> rating	<small>123</small> age
1	[NULL]	[NULL]	[NULL]	29	Brutus	1	33
2	[NULL]	[NULL]	[NULL]	32	Andy	8	25,5
3	[NULL]	[NULL]	[NULL]	58	Rusty	10	35
4	[NULL]	[NULL]	[NULL]	71	Zorba	10	16
5	[NULL]	[NULL]	[NULL]	85	Art	3	22,5
6	[NULL]	[NULL]	[NULL]	95	Bob	3	63,5

# JOIN(s) e ANTI JOIN(s) con RELATIVA QUERY

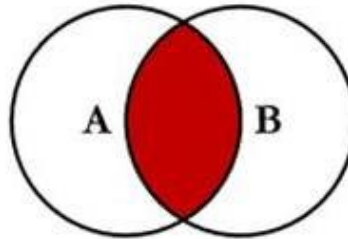
## SQL JOINS



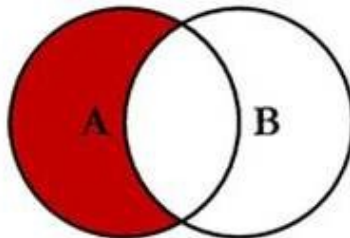
```
SELECT <select_list>  
FROM TableA A  
LEFT JOIN TableB B  
ON A.Key = B.Key
```



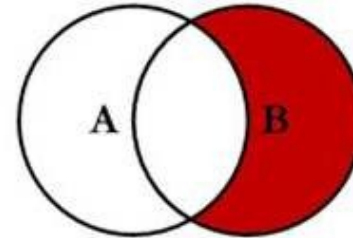
```
SELECT <select_list>  
FROM TableA A  
RIGHT JOIN TableB B  
ON A.Key = B.Key
```



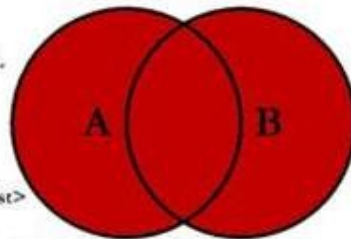
```
SELECT <select_list>  
FROM TableA A  
INNER JOIN TableB B  
ON A.Key = B.Key
```



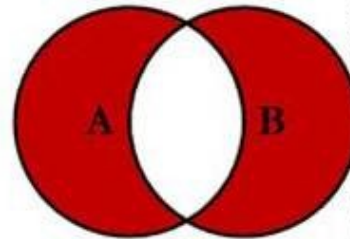
```
SELECT <select_list>  
FROM TableA A  
LEFT JOIN TableB B  
ON A.Key = B.Key  
WHERE B.Key IS NULL
```



```
SELECT <select_list>  
FROM TableA A  
RIGHT JOIN TableB B  
ON A.Key = B.Key  
WHERE A.Key IS NULL
```



```
SELECT <select_list>  
FROM TableA A  
FULL OUTER JOIN TableB B  
ON A.Key = B.Key
```



```
SELECT <select_list>  
FROM TableA A  
FULL OUTER JOIN TableB B  
ON A.Key = B.Key  
WHERE A.Key IS NULL  
OR B.Key IS NULL
```

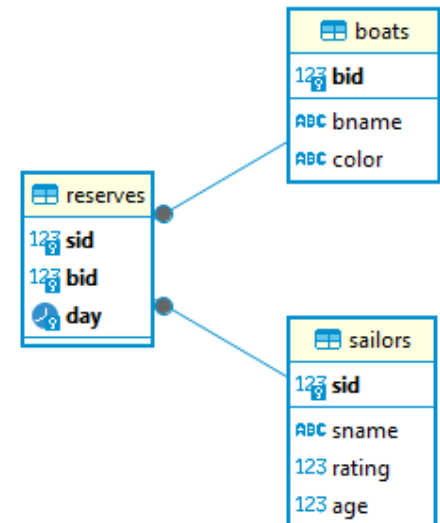


# **LINEE GUIDA PRATICHE PER PROGETTARE QUERY SQL**

# Studia lo schema

R1: accertati di aver compreso bene lo schema del database che devi interrogare!

- Se disponibile, consulta il diagramma con lo schema relazionale in forma grafica
- Memorizza i nomi delle tabelle e dei loro attributi (per quanto possibile)
- Verifica il dominio di ogni attributo (per capire quali operazioni potrai applicare a ognuno di essi)
- Accertati di aver compreso bene i vincoli di integrità referenziale (ti serviranno per i JOIN)
- Se necessario, prova a scrivere in linguaggio naturale che cosa rappresenta ogni tabella



# Leggi bene la richiesta!

R2: Leggi con molta attenzione la formulazione della query in linguaggio naturale (Italiano, Inglese, ...)

- Elenca gli attributi che sono richiesti nell'output (□ *target-list*)
- Associa ognuno di questi attributi alla tabella dove compaiono (□ costruisci la *relation-list*)
- Identifica i filtri che dovranno essere applicati per eliminare le tuple indesiderate

# Cerca le aggregazioni!

R3: Prova a identificare parole o espressioni che suggeriscono l'uso di funzioni di aggregazione:

- Restituire il valore medio ... □ AVG?
- Trovare il totale ... □ SUM?
- Il numero di clienti che ... □ COUNT?
- Il fatturato annuale diviso per venditore ... □ SUM + GROUP BY?
- I venditori il cui fatturato annuale è superiore a 20.000 Euro ... □ SUM + GROUP BY + HAVING?

# Comprendi i vincoli di IR

R4: Identifica con precisione le condizioni che dovrai usare per fare il JOIN di due o più tabelle

- Identifica gli attributi che dovrai utilizzare (tipicamente chiave primaria □ chiave esterna)
- Ricorda che il WHERE è una funzione che si applica tupla per tupla (e non su più tuple contemporaneamente)
- Rifletti con attenzione sul tipo di JOIN che dovrai applicare per esser semanticamente fedele alla richiesta (INNER, OUTER LEFT, OUTER RIGHT, ANTI JOIN, ...)

# Cerca le query annidate!

R5: Identifica se la query rientra in uno dei casi d'uso per la costruzione di query annidate

- Devo fare un confronto tra il valore di un attributo in una tupla e un valore calcolato dinamicamente da una sub query
- Devo verificare se il valore di un attributo in una tupla appartiene a un insieme di valori □ IN
- Devo confrontare un valore con un insieme di valori generati da una sub query □ ANY, ALL
- Devo verificare se il valore di un attributo in una sub query correlata produce almeno una tupla come risultato □ EXISTS

# Trova eventuali operazioni insiemistiche

R5: Ove è il caso, usa diagrammi (tipo Venn) per determinare le operazioni insiemistiche di cui hai bisogno nella query

- Verifica se devi unire tuple con lo stesso schema ma che provengono da tabelle diverse (□ UNION)
- Verifica se devi trovare tuple comuni tra insiemi di tuple che provengono da diverse tabelle con lo stesso schema (□ INTERSECT)
- Verifica se devi trovare tuple che compaiono in un insieme ma non in un altro (□ EXCEPT)
- ...

# Verifica la coerenza (semantica) dei risultati

R6: Verifica che intuitivamente il risultato della query sia quello che corrisponde alla richiesta

- Scomponi la query in query più semplici per verificare che ogni «pezzo» della query restituisca quello che ti aspettavi
- Verifica che non ci siano valori che violano qualche filtro (es. se chiedi i marinai con rating  $> 5$ , non devono comparire marinai che nella tabella Sailors hanno valore 4 )
- Verifica che il numero di tuple restituite sia intuitivamente della cardinalità corretta
- Controlla di aver restituito tutti e soli i campi che erano richiesti nella query
- Verifica di non aver dimenticato di riordinare le tuple di output (se richiesto)
- Verifica se hai usato il DISTINCT dove è necessario





# Esempi di query di base

Q0: sname ed età dei marinai

```
select s.sname, s.age  
from sailors s;
```



	ABC sname ▼	123 age ▼
1	Dustin	45
2	Brutus	33
3	Iubber	55,5
4	Andy	25,5
5	Rusty	35
6	Horatio	35
7	Zorba	16
8	Horatio	35
9	Art	22,5
10	Bob	63,5

Q0.1 sname ed età dei marinai  
che hanno più di 50 anni

```
select s.sname, s.age  
from sailors s  
where s.age > 50;
```

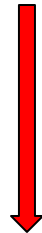


	ABC sname ▼	123 age ▼
1	Iubber	55,5
2	Bob	63,5

# Esempi di query di base

(Q1) Trovare i nomi dei marinai che hanno prenotato la barca #103

```
SELECT S.sname  
FROM   Sailors S, Reserves R  
WHERE  S.sid=R.sid AND R.bid=103
```



	ABC sname ▼
1	Dustin
2	Iubber
3	Horatio

# Esempi

sid di marinai che hanno noleggiato una barca rossa

```
SELECT R.sid
FROM Boats B, Reserves R
WHERE B.bid = R.bid AND B.color = 'red'
```



	123 sid
1	22
2	31
3	64
4	22
5	31

sname di marinai che hanno noleggiato una barca rossa

```
SELECT S.sname
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid AND B.color = 'red'
```

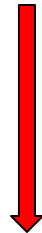


	ABC sname
1	Dustin
2	lubber
3	Horatio
4	Dustin
5	lubber

# Esempi

## Uso di espressioni aritmetiche e pattern nelle query

```
SELECT S.age, age1=S.age-5, 2*S.age AS age2  
FROM Sailors S  
WHERE S.sname LIKE 'B_%B'
```




	123 age ▼	123 age1 ▼	123 age2 ▼
1	63,5	58,5	127

# Esempi

(Q6) Nomi di marinai che hanno prenotato sia una barca rossa, sia una barca verde

```
SELECT DISTINCT s.sid
from Sailors s , Boats b1 , Reserves r1 , Boats b2, Reserves r2
where s.sid = r1.sid and r1.bid = b1.bid and
s.sid = r2.sid and r2.bid = b2.bid
and b1.color = 'red' and b2.color = 'green';
```

```
SELECT s.sid from Sailors s , Boats b , Reserves r
where s.sid = r.sid and r.bid = b.bid and b.color = 'red'
INTERSECT
SELECT sl.sid from Sailors sl , Boats bl , Reserves rl
where sl.sid = rl.sid and rl.bid = bl.bid and bl.color = 'green';
```




	123 sid ▼
1	22
2	31

# Esempi

(Q19) Trovare i sid dei marinai che hanno prenotato barche rosse ma non barche verdi

```
SELECT r.sid from Boats b , Reserves r
where r.bid = b.bid and b.color = 'red'
EXCEPT
SELECT r.sid from Boats b , Reserves r
where r.bid = b.bid and b.color = 'green';
```



	123 sid ▼
1	64