

Ripasso finale su SQL

# Overview programma (pt.1)

## **INTRODUZIONE AL CORSO** (slide1)

Il sistema informativo nell'azienda

Caratteristiche dei DBMS e concetto di transazione (slide8)

Utenti, strumenti e moduli di un DBMS

## **TEORIA RELAZIONALE**

Il modello relazionale dei dati, sua definizione (informale e formale) e concetti fondamentali (slide2)

Algebra relazionale: operazioni unarie e binarie (slide3)

Interrogazioni in algebra relazionale e loro ottimizzazione (slide3)

Calcolo relazionale: definizione, equivalenza fra algebra relazionale e calcolo relazionale (slide4)

Datalog: definizione, interrogazioni ricorsive, specifica di vincoli di integrità (slide4)

Relazioni tra i poteri espressivi dei linguaggi formali presentati (slide3-4)

# Overview programma (pt.2)

## LINGUAGGI PER BASI DI DATI

Introduzione a SQL: standardizzazione di SQL, domini SQL, vincoli e semplice definizione di tabelle ([slide5](#))

SQL come DDL: integrità referenziale, definizione di schemi e loro modifica, cataloghi relazionali ([slide5](#))

Interrogazioni SQL semplici, interrogazioni con ordinamenti e raggruppamenti, Interrogazioni SQL complesse ([slide6-7](#))

Comandi di modifica ([slide5](#)) e viste in SQL ([slide7](#))

Aspetti evoluti del DDL: indici ([slide11](#)) e vincoli di integrità generici ([slide8](#)), controllo dell'accesso ([slide7](#)), viste e controllo dell'accesso ([slide7](#)), trigger ([slide8](#)), procedure ([slide8](#)), transazioni ([slide8](#)) (cenni)

## PROGETTAZIONE DI BASI DI DATI

Fasi della progettazione ([slide9](#))

Le astrazioni nella progettazione dei dati ([slide9](#))

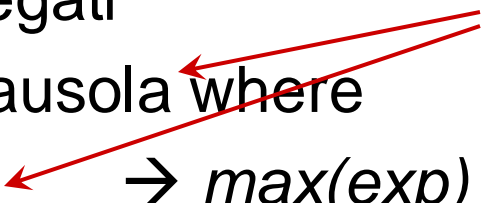
Il modello Entità-Relazioni: definizione, associazioni, identificatori e gerarchie, proprietà delle gerarchie ([slide9](#))

Il ciclo di progettazione: fasi di progettazione, strategie, qualità di un progetto concettuale ([slide10](#))

Progettazione logica: fasi del progetto logico, eliminazione delle gerarchie, gestione degli identificatori, gestione degli attributi, gestione delle associazioni. ([slide10](#))

Normalizzazione (cenni). Progettazione fisica (cenni) ([slide11](#))

# Cose che **non** vogliamo vedere (pt. 1)

- Target list **miste** quando non c'è la clausola group by
- Attributi nella select o nella having che non siano **univoci** rispetto alla clausola group by (quando c'è una clausola group by)
- Aggregati di aggregati
- Aggregati nella clausola where
- HAVING max(X).  *→ max(exp) non è un predicato !*
- HAVING count(\*) = 0 *→ se c'è il gruppo... non è vuoto !*
- count( exp<sub>1</sub> = exp<sub>2</sub> ) *→ si contano le tuple, non i predicati !*

# Cose che **non** vogliamo vedere (pt.2)

- Clausole where auto-contraddittorie
- IN / NOT IN con
  - Niente a sinistra `where [not] in ( select ... )`
  - Schemi che non si corrispondono  
`where DataInizio [not] in ( select Nome, Cognome ... )`
- Da usare con consapevole parsimonia (*non sono errori*):
  - Predicati con query a destra e a sinistra
  - Predicati con query nidificate a destra senza ANY/ALL
  - EXISTS (SELECT X...) con X != \*

# Interpretazione

- ```
select CodCli
from Ordine O
where exists (select *
              from Ordine O1
              where O1.CodCli = O.CodCli
                 and O1.Data = O.Data
                 and O1.CodOrd <> O.CodOrd)
```
- Per ogni tupla di Ordine:
  - Si passano alla query interna CodCli, Data, CodOrd
  - Si valuta la query interna
  - Se la query non è vuota il CodCli di quella tupla è proiettato nel risultato

# Equivalenza del potere espressivo

- IN, =ANY, EXISTS hanno lo stesso potere espressivo e possono essere espresse tramite join (a meno di duplicati)
- NOT IN, <>ALL, NOT EXISTS hanno lo stesso potere espressivo e possono essere rese tramite la differenza
- <comp> ANY, a meno di duplicati, può essere reso con un theta-join (non equi-join)
- <comp> ALL può essere reso con query che abbiano raggruppamento e estrazione di minimo o massimo

# CONVEGNI SCIENTIFICI

Si consideri il seguente schema di base di dati, che rappresenta le iscrizioni a convegni scientifici.

CONVEGNO (Serie, Anno, Nazione, DataInizio, DataFine, Capienza)

UTENTE (CodUtente, Nome, Cognome, Nazione)

TIPOLISCRIZIONE (Serie, Anno, NomeTipo, Prezzo)

ISCRIZIONE (CodUtente, Serie, Anno, NomeTipoliscrizione, DataIscrizione)

Ogni convegno è identificato dal nome della serie (es. “Very Large Databases”) e dall’anno di edizione. Ogni convegno può supportare diversi tipi di iscrizione a diversi prezzi (es. iscrizione “early”, “regular” o “late”). L’attributo *Capienza* della relazione CONVEGNO rappresenta il numero massimo di iscrizioni che possono essere accettate per quel convegno. L’attributo *CodUtente* nelle varie relazioni è un valore intero.



CONVEGNO (Serie, Anno, Nazione, DataInizio, DataFine, Capienza)

UTENTE (CodUtente, Nome, Cognome, Nazione)

TIPOLISCRIZIONE (Serie, Anno, NomeTipo, Prezzo)

ISCRIZIONE (CodUtente, Serie, Anno, NomeTipoliscrizione, DataIscrizione)

Specificare in SQL la creazione della tabella ISCRIZIONE, definendo i vincoli di tupla e di dominio ritenuti opportuni ed esprimendo eventuali vincoli di integrità referenziale relativi a tutte le tabelle dello schema.

```
CREATE TABLE Iscrizione (  
    CodUtente INTEGER REFERENCES Utente(CodUtente)  
        ON UPDATE CASCADE ON DELETE NO ACTION,  
    Serie VARCHAR(255),  
    Anno INTEGER,  
    NomeTipoliscrizione NOT NULL VARCHAR(255),  
    DataIscrizione NOT NULL DATE,  
    PRIMARY KEY (CodUtente, Serie, Anno),  
    FOREIGN KEY (Serie, Anno, NomeTipoliscrizione)  
        REFERENCES Tipoliscrizione (Serie, Anno, NomeTipo)  
        ON UPDATE CASCADE ON DELETE NO ACTION  
)
```

CONVEGNO (Serie, Anno, Nazione, DataInizio, DataFine, Capienza)

UTENTE (CodUtente, Nome, Cognome, Nazione)

TIPOLISCRIZIONE (Serie, Anno, NomeTipo, Prezzo)

ISCRIZIONE (CodUtente, Serie, Anno, NomeTipoliscrizione, DataIscrizione)

Specificare in SQL la creazione della tabella ISCRIZIONE, definendo i vincoli di tupla e di dominio ritenuti opportuni ed esprimendo eventuali vincoli di integrità referenziale relativi a tutte le tabelle dello schema.

**CREATE TABLE** Iscrizione (

CodUtente **INTEGER REFERENCES** Utente(CodUtente)

**ON UPDATE CASCADE ON DELETE NO ACTION,**

Serie **VARCHAR**(255),

Anno **INTEGER**,

NomeTipoliscrizione **NOT NULL VARCHAR**(255),

DataIscrizione **NOT NULL DATE** **check** (DataIscrizione <= (SELECT DataFine

FROM Convegno C

WHERE (C.Serie,C.Anno) = (Serie,Anno))),

**PRIMARY KEY** (CodUtente, Serie, Anno),

**FOREIGN KEY** (Serie, Anno, NomeTipoliscrizione)

**REFERENCES** Tipoliscrizione (Serie, Anno, NomeTipo)

**ON UPDATE CASCADE ON DELETE NO ACTION**

Possiamo aggiungere vincolo che impone che non ci siano iscrizioni successive alla convegno

CONVEGNO (Serie, Anno, Nazione, DataInizio, DataFine, Capienza)

UTENTE (CodUtente, Nome, Cognome, Nazione)

TIPOLISCRIZIONE (Serie, Anno, NomeTipo, Prezzo)

ISCRIZIONE (CodUtente, Serie, Anno, NomeTipoliscrizione, DataIscrizione)

Trovare il/i convegno/i con il ricavo più alto.

```
CREATE VIEW RicavoConv(Serie, Anno, Ricavo) AS (  
    SELECT T.Serie, T.Anno, SUM(T.Prezzo)  
    FROM Iscrizione AS I, Tipoliscrizione AS T  
    WHERE I.Serie=T.Serie AND I.Anno=T.Anno AND I.NomeTipoliscrizione=T.NomeTipo  
    GROUP BY T.Serie, T.Anno  
)
```

```
SELECT Serie, Anno  
FROM RicavoConv  
WHERE Ricavo = (  
    SELECT MAX(Ricavo)  
    FROM RicavoConv  
)
```

CONVEGNO (Serie, Anno, Nazione, DataInizio, DataFine, Capienza)

UTENTE (CodUtente, Nome, Cognome, Nazione)

TIPOLISCRIZIONE (Serie, Anno, NomeTipo, Prezzo)

ISCRIZIONE (CodUtente, Serie, Anno, NomeTipoliscrizione, DataIscrizione)

Trovare codice, cognome e nome degli utenti che non si sono mai iscritti a convegni svolti nella loro nazione.

-- versione1 (not in)

**SELECT** U.CodUtente, U.Cognome, U.Nome

**FROM** Utente **AS** U

**WHERE** U.CodUtente **NOT IN** (

**SELECT** I.CodUtente

**FROM** Iscrizione **AS** I, Convegno **AS** C

**WHERE** I.Serie=C.Serie **AND** I.Anno=C.Anno **AND** C.Nazione=U.Nazione)

-- versione2 (not exists)

**SELECT** U.CodUtente, U.Cognome, U.Nome

**FROM** Utente **AS** U

**WHERE NOT EXISTS** (SELECT \*

**FROM** Iscrizione **AS** I, Convegno **AS** C

**WHERE** I.Serie=C.Serie **AND** I.Anno=C.Anno **AND** C.Nazione=U.Nazione  
**AND** U.CodUtente = I.CodUtente)

CONVEGNO (Serie, Anno, Nazione, DataInizio, DataFine, Capienza)

UTENTE (CodUtente, Nome, Cognome, Nazione)

TIPOLISCRIZIONE (Serie, Anno, NomeTipo, Prezzo)

ISCRIZIONE (CodUtente, Serie, Anno, NomeTipoliscrizione, DataIscrizione)

Trovare le serie che nel 2016 hanno incrementato il numero di iscritti almeno del 20% rispetto al 2015.

```
SELECT I.Serie  
FROM Iscrizione AS I  
WHERE I.Anno=2016  
GROUP BY I.Serie  
HAVING COUNT(*) >= 1.2 * (
```

```
SELECT COUNT(*)  
FROM Iscrizione AS I2  
WHERE I2.Serie=I.Serie AND I2.Anno=2015  
)
```

CONVEGNO (Serie, Anno, Nazione, DataInizio, DataFine, Capienza)

UTENTE (CodUtente, Nome, Cognome, Nazione)

TIPOLISCRIZIONE (Serie, Anno, NomeTipo, Prezzo)

ISCRIZIONE (CodUtente, Serie, Anno, NomeTipoliscrizione, DataIscrizione)

Specificare in SQL il vincolo che verifica che ogni convegno abbia un numero di iscritti che non eccede la sua capienza. (2 punti)

```
CREATE ASSERTION VerificaCapienza CHECK (  
    NOT EXISTS (  
        SELECT C.Serie, C.Anno  
        FROM Iscrizione AS I, Convegno AS C  
        WHERE I.Serie=C.Serie AND I.Anno=C.Anno  
        GROUP BY C.Serie, C.Anno, C.Capienza  
        HAVING COUNT(*)>C.Capienza  
    )  
)
```

*/\* Notare che la Capienza può essere utilizzata in forma non aggregata nella clausola having perché è unica nel gruppo (qui è stata aggiunta negli attributi di raggruppamento ma può essere omessa, dato che è funzionalmente determinata dai due precedenti \*/*

# HOTEL DI LUSSO

## Versione schema 1

Il seguente schema rappresenta i dati relativi ai pernottamenti già conclusi dei clienti di un albergo aperto continuativamente dal 2004. **Per semplicità, di ogni pernottamento si registrano i dati completi di uno solo degli occupanti della camera.**

Le tuple nella tabella SOGGIORNO vengono sempre inserite all'inizio del soggiorno, lasciando gli ultimi due attributi NULL. Gli attributi vengono poi aggiornati con i valori corretti quando i clienti fanno il check out.

CLIENTE(CodiceFiscale, Nome, Indirizzo, Telefono, Email)

CAMERA(Numero, Piano, Tipo, PostiLetto, NumLettiSingoli, NumLettiMatr, Superficie, Descr)

SOGGIORNO(**CodiceFiscale**, NumCamera, DataCkIn, DataCkOut, CostoTotale)

MINIBAR(NumCamera, Data, ArticoloConsumato, Quantità)

E' appropriato inserire CodiceFiscale nella chiave di SOGGIORNO?

## Versione schema 2 (utilizzata nei seguenti esercizi)

Il seguente schema rappresenta i dati relativi ai pernottamenti già conclusi dei clienti di un albergo aperto continuativamente dal 2004. Per semplicità, di ogni pernottamento si registrano i dati completi di uno solo degli occupanti della camera.

Le tuple nella tabella SOGGIORNO vengono sempre inserite all'inizio del soggiorno, lasciando gli ultimi due attributi NULL. Gli attributi vengono poi aggiornati con i valori corretti quando i clienti fanno il check out.

CLIENTE(CodiceFiscale, Nome, Indirizzo, Telefono, Email)

CAMERA(Numero, Piano, Tipo, PostiLetto, NumLettiSingoli, NumLettiMatr, Superficie, Descr)

SOGGIORNO(NumCamera, DataCkIn, DataCkOut, CodiceFiscale, CostoTotale)

MINIBAR(NumCamera, Data, ArticoloConsumato, Quantità)



CLIENTE(CodiceFiscale, Nome, Indirizzo, Telefono, Email)

CAMERA(Numero, Piano, Tipo, PostiLetto, NumLettiSingoli, NumLettiMatr, Superficie, Descr)

SOGGIORNO(NumCamera, DataCkIn, DataCkOut, CodiceFiscale, CostoTotale)

MINIBAR(NumCamera, Data, ArticoloConsumato, Quantità)

Scrivere il comando SQL per creare la tabella SOGGIORNO, effettuando opportune e ragionevoli ipotesi su domini, vincoli e reazioni ai cambiamenti.

Create table SOGGIORNO (

NumCamera integer references CAMERA(NumCamera)

on delete no action on update cascade,

DataCkIn Date,

DataCkOut Date

check(DataCkOut NOT NULL AND DataCkOut > DataCkIn OR DataCkOut IS NULL),

CodiceFiscale char(16) references CLIENTE(CodiceFiscale)

on delete no action on update cascade,

CostoTotale decimal(8,2) > 0

PRIMARY KEY (NumCamera,DataCkIn)

)

CLIENTE(CodiceFiscale, Nome, Indirizzo, Telefono, Email)

CAMERA(Numero, Piano, Tipo, PostiLetto, NumLettiSingoli, NumLettiMatr, Superficie, Descr)

SOGGIORNO(NumCamera, DataCkIn, DataCkOut, CodiceFiscale, CostoTotale)

MINIBAR(NumCamera, Data, ArticoloConsumato, Quantità)

Scrivere il comando SQL per creare la tabella MINIBAR, effettuando opportune e ragionevoli ipotesi su domini, vincoli e reazioni ai cambiamenti.

Create table MINIBAR (

NumCamera integer references CAMERA(NumCamera)

on delete no action on update cascade,

Data Date check(EXISTS (SELECT \* FROM Soggiorno S

WHERE NumCamera = S.NumCamera

AND (S.DataCkOut IS NOT NULL

AND Data between S.DataCkIn and S.DataCkOut

OR S.DataCkOut IS NULL AND Data >= S.DataCkIn)

ArticoloConsumato varchar(100),

Quantita integer >0,

Primary Key (NumCamera, Data, ArticoloConsumato)

)

*check che un Soggiorno corrispondente deve esistere nella corrispondente Data e NumCamera. Si è complicato per tener conto del fatto che DataCkOut può essere null*

*Notare che, per fare la join correttamente tra queste due tabelle, richiedere l'uguaglianza del numero della stanza non è sufficiente. Anche la Date deve cadere nell'intervallo corrispondente del soggiorno.*

CLIENTE(CodiceFiscale, Nome, Indirizzo, Telefono, Email)

CAMERA(Numero, Piano, Tipo, PostiLetto, NumLettiSingoli, NumLettiMatr, Superficie, Descr)

SOGGIORNO(NumCamera, DataCkIn, DataCkOut, CodiceFiscale, CostoTotale)

MINIBAR(NumCamera, Data, ArticoloConsumato, Quantità)

Costruire la classifica dei prodotti più prelevati dai minibar nel 2018 (dai più graditi ai meno graditi).

```
select ArticoloConsumato,  
       sum(Quantità) as ItemTotaliConsumati  
from Minibar  
where Data between 1/1/2018 and 31/12/2018  
group by ArticoloConsumato  
order by 2 desc
```

CLIENTE(CodiceFiscale, Nome, Indirizzo, Telefono, Email)

CAMERA(Numero, Piano, Tipo, PostiLetto, NumLettiSingoli, NumLettiMatr, Superficie, Descr)

SOGGIORNO(NumCamera, DataCkIn, DataCkOut, CodiceFiscale, CostoTotale)

MINIBAR(NumCamera, Data, ArticoloConsumato, Quantità)

Estrarre i nomi dei clienti che hanno consumato un Chinotto durante il loro primo soggiorno in albergo.

**SOL1:**

**select Nome**

**from Minibar natural join Soggiorno **S** natural join Cliente**

**where ArticoloConsumato = «Chinotto»**

**and Data between DataCkIn and DataCkOut**

**and DataCkIn = (**

**select min(DataCkIn)**

**from Soggiorno**

**where CodiceFiscale = **S**. CodiceFiscale**

**)**

CLIENTE(CodiceFiscale, Nome, Indirizzo, Telefono, Email)

CAMERA(Numero, Piano, Tipo, PostiLetto, NumLettiSingoli, NumLettiMatr, Superficie, Descr)

SOGGIORNO(NumCamera, DataCkIn, DataCkOut, CodiceFiscale, CostoTotale)

MINIBAR(NumCamera, Data, ArticoloConsumato, Quantità)

Estrarre i nomi dei clienti che hanno consumato un Chinotto durante il loro primo soggiorno in albergo.

SOL2:

```
create view PrimoSoggiorno(CFCli, NumCam, PrimoCkIn, PrimoCkOut) as
```

```
select CodiceFiscale, NumCamera, DataCkIn, DataCkOut
```

```
from Soggiorno
```

```
where (CodiceFiscale, DataCkIn) = (select CodiceFiscale, min(DataCkIn)
```

```
from Soggiorno
```

```
group by CodiceFiscale )
```

```
select Nome
```

```
from (Minibar natural join PrimoSoggiorno) join Customer on CFCli = CodiceFiscale
```

```
where ArticoloConsumato = «Chinotto»
```

```
and Data between PrimoCkIn and PrimoCkOut
```

CLIENTE(CodiceFiscale, Nome, Indirizzo, Telefono, Email)

CAMERA(Numero, Piano, Tipo, PostiLetto, NumLettiSingoli, NumLettiMatr, Superficie, Descr)

SOGGIORNO(NumCamera, DataCkIn, DataCkOut, CodiceFiscale, CostoTotale)

MINIBAR(NumCamera, Data, ArticoloConsumato, Quantità)

Di ciascun cliente che ha speso complessivamente più di 10.000 euro (in tutti e 18 gli anni di apertura) si indichi il numero totale di notti trascorse in albergo in soggiorni “recenti”, cioè iniziati dopo il 1/1/2021.

```
select CodiceFiscale,  
       sum(DataCkOut – DataCkIn) as NottiRecentiDaNoi  
from Soggiorno  
where DataCkIn > 1/1/2021  
      and CodiceFiscale in ( select CodiceFiscale  
                           from Soggiorno  
                           group by CodiceFiscale  
                           having sum( CostoTotale ) > 10.000 )  
group by CodiceFiscale
```

CLIENTE(CodiceFiscale, Nome, Indirizzo, Telefono, Email)

CAMERA(Numero, Piano, Tipo, PostiLetto, NumLettiSingoli, NumLettiMatr, Superficie, Descr)

SOGGIORNO(NumCamera, DataCkIn, DataCkOut, CodiceFiscale, CostoTotale)

MINIBAR(NumCamera, Data, ArticoloConsumato, Quantità)

Esprimere il vincolo per cui non possono esserci consumazioni dal minibar per camere non occupate

```
create assertion NessunoSnackPerFantasmi as check (  
  not exists (  
    select *  
    from Minibar M  
    where not exists (  
      select *  
      from Soggiorno  
      where NumCamera = M.NumCamera and M.Data between DataCkIn and DataCkOut  
    )  
  )  
)
```

```
create assertion NessunoSnackPerFantasmi as check (  
  not exists (  
    select *  
    from Minibar M  
    where NumCamera not in ( select NumCamera  
                             from Soggiorno  
                             where M.Data between DataCkIn and DataCkOut  
    )  
  )  
)
```

# AUTO ELETTRICHE

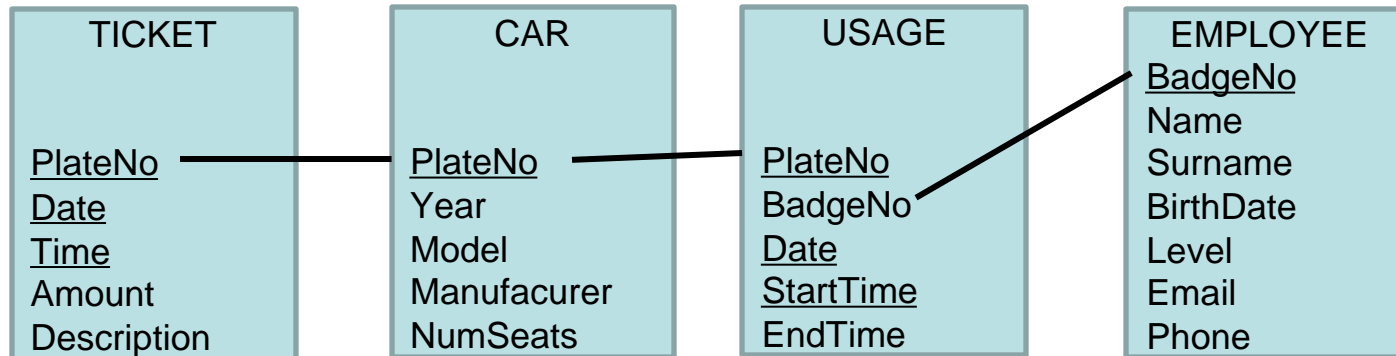
CAR ( PLATENo, YEAR, MODEL, MANUFACTURER, NUMSEATS )

EMPLOYEE ( BADGENo, NAME, SURNAME, BIRTHDATE, LEVEL, EMAIL, PHONE )

USAGE ( PLATENo, BADGENo, DATE, STARTTIME, ENDTIME )

TICKET ( PLATENo, DATE, TIME, AMOUNT, DESCRIPTION )

Lo schema sopra descrive l'utilizzo di auto elettriche degli impiegati di un'azienda di Milano, per spostamenti giornalieri durante le ore di ufficio. Le auto sono ritirate nel garage dell'azienda avvicinando il badge al lettore sulla porta e vengono considerate restituite quando un sensore le registra al rientro nel garage. Se i guidatori violano regole del codice della strada e ricevono delle multe mentre usano l'auto, l'ammontare corrispondente viene sottratto direttamente dal loro stipendio successive.





CAR ( PLATENo, YEAR, MODEL, MANUFACTURER, NUMSEATS )

EMPLOYEE (BADGENo, NAME, SURNAME, BIRTHDATE, LEVEL, EMAIL, PHONE )

USAGE ( PLATENo, BADGENo, DATE, STARTTIME, ENDTIME )

TICKET ( PLATENo, DATE, TIME, AMOUNT, DESCRIPTION )

1. Scrivere le istruzioni di creazione delle tabelle **USAGE** e **TICKET**, definendo ragionevoli vincoli di tupla e di dominio ed esprimendo opportuni vincoli di integrità referenziale (foreign keys) verso altre tabelle.

CAR ( PLATENo, YEAR, MODEL, MANUFACTURER, NUMSEATS )

EMPLOYEE (BADGENo, NAME, SURNAME, BIRTHDATE, LEVEL, EMAIL, PHONE )

USAGE ( PLATENo, BADGENo, DATE, STARTTIME, ENDTIME )

TICKET ( PLATENo, DATE, TIME, AMOUNT, DESCRIPTION )

2. Estrarre Name e Surname dell'impiegato che, nel 2018, ha ricevuto multe con l'ammontare massimo di tutta l'azienda

```
select Name, Surname from Employee
where BadgeNo in (
    select BadgeNo
    from USAGE U join TICKET T on ( U.PlateNo, U.Date ) = ( T.PlateNo, T.Date )
    where year(U.Date) = 2018 and T.Time between U.StartTime and U.EndTime
    group by BadgeNo
    having sum(Amount) >= ALL
        ( select sum(Amount)
          from USAGE U join TICKET T on ( U.PlateNo,U.Date ) = ( T.PlateNo, T.Date )
          where year(U.Date) = 2018 and T.Time between U.StartTime and U.EndTime
          group by BadgeNo ) )
```



CAR ( PLATENo, YEAR, MODEL, MANUFACTURER, NUMSEATS )

EMPLOYEE (BADGENo, NAME, SURNAME, BIRTHDATE, LEVEL, EMAIL, PHONE )

USAGE ( PLATENo, BADGENo, DATE, STARTTIME, ENDTIME )

TICKET ( PLATENo, DATE, TIME, AMOUNT, DESCRIPTION )

3. Estrarre gli impiegati di livello “Product Manager” che non hanno mai utilizzato la stessa auto due volte.

*Restringendo la selezione agli impiegati che hanno guidato almeno una volta*

```
select BadgeNo  
from EMPLOYEE natural join USAGE  
where Level = "Product Manager"  
group by BadgeNo  
having count(*) = count( distinct PlateNo )
```

*Includendo anche gli impiegati che non hanno mai guidato una macchina:*

```
select *  
from EMPLOYEE  
where Level = "Product Manager"  
and BadgeNo not in ( select BadgeNo  
                      from USAGE  
                      group by BadgeNo, PlateNo  
                      having count(*) > 1 )
```

*Con un outer join possiamo includere anche i product manager che non guidano, seguendo lo «stile» della soluzione precedente:*

```
select BadgeNo
from EMPLOYEE E left join USAGE U on E.BadgeNo = U.BadgeNo
where Level = "Product Manager"
group by E.BadgeNo
having count( PlateNo ) = count( distinct PlateNo )
```



CAR ( PLATENo, YEAR, MODEL, MANUFACTURER, NUMSEATS )

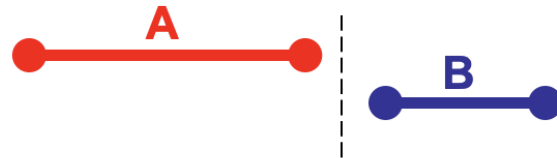
EMPLOYEE (BADGENo, NAME, SURNAME, BIRTHDATE, LEVEL, EMAIL, PHONE )

USAGE ( PLATENo, BADGENo, DATE, STARTTIME, ENDTIME )

TICKET ( PLATENo, DATE, TIME, AMOUNT, DESCRIPTION )

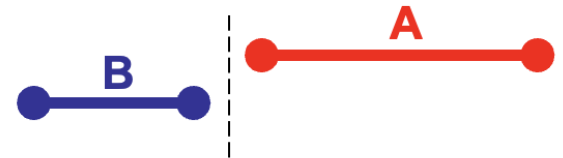
4. Esprimere il vincolo che controlla che non ci sono due utilizzi distinti di una stessa auto che si sovrappongano nel tempo

***Intervalli disgiunti:***



$A.End < B.Start$

OR



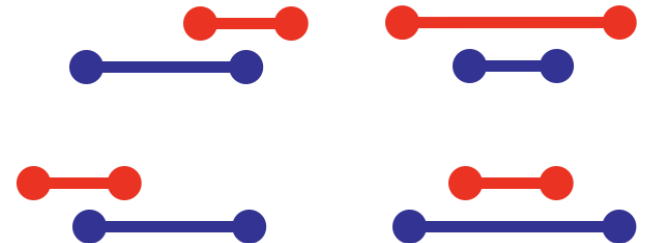
$B.End < A.Start$

***Intervalli sovrapposti =***

NOT ***disgiunti*** =

NOT (  $A.End < B.Start$  OR  $B.End < A.Start$  ) =

$A.End \geq B.Start$  AND  $B.End \geq A.Start$



```
create assertion NoOverlap as check(
  not exists (
    select *
    from USAGE A join USAGE B
      on A.PlateNo = B.PlateNo
    where A.Date = B.Date
    and A.StartTime > B.StartTime
    and A.EndTime >= B.StartTime
    and B.EndTime >= A.StartTime  ) )
```

Questa è una soluzione più performante rispetto a  
and A.StartTime <> B.StartTime  
dato che si dimezzano i casi da considerare, senza ledere la generalità

**Discussione** su necessità di aggiungere una condizione che impedisca di confrontare uno Usage con se stesso.

In casi simili, si impone una disuguaglianza sull'attributo che identifica la tabella. In questo caso la chiave primaria è composta da tre attributi: PlateNo, Date, e StartTime. Basterebbe chiedere che almeno uno dei tre fosse diverso, ma siccome A.PlateNo = B.PlateNo e A.Date = B.Date per la richiesta dell'esercizio, l'unica possibilità è quella di imporre la disuguaglianza tra gli orari di inizio degli intervalli.

Qualcuno potrebbe obiettare che, così facendo, l'assertion non permetterebbe di escludere l'inserimento di, ad esempio, ('AB123CD', 1234, 2021-01-01, 15:00, 16:00) e ('AB123CD', 1234, 2021-01-01, 15:00, 18:00), ma questa evenienza è già esclusa dall'unicità della chiave primaria, che non permette inserimento di tuple con pari targa, data e ora di inizio.

Alternativa che sfrutta la simmetria:

```
create assertion NOOVERLAP as check(  
  not exists ( select *  
               from USAGE A join USAGE B  
                 on A.PlateNo = B.PlateNo  
                where A.Date = B.Date  
                  and A.StartTime <> B.StartTime  
                  and A.StartTime between B.StartTime and B.EndTime) )
```

CAR ( PLATENo, YEAR, MODEL, MANUFACTURER, NUMSEATS )

EMPLOYEE (BADGENo, NAME, SURNAME, BIRTHDATE, LEVEL, EMAIL, PHONE )

USAGE ( PLATENo, BADGENo, DATE, STARTTIME, ENDTIME )

TICKET ( PLATENo, DATE, TIME, AMOUNT, DESCRIPTION )

5. Calcolare, per ogni auto, il numero medio di utilizzi giornalieri e la durata media di questi utilizzi. **Si consideri che le giornate in cui una specifica auto non è stata utilizzata non devono contribuire al calcolo della media (non della durata media, ma solo al numero di utilizzi, contando quel giorno). Inoltre, per semplicità, si assuma che la differenza tra due timestamps ritorni direttamente un intervallo.**

**(a)**

```
select PlateNo,  
       count(*)/count(distinct Date) as AvgNumOfDailyUses,  
       avg(EndTime - StartTime) as AvgUseDuration  
from USAGE  
group by PlateNo
```

*Questa non è l'interpretazione corretta, dato che considera solo, per ogni auto, i giorni in cui quell'auto è stata utilizzata almeno una volta, e non –invece– i giorni in cui l'auto non è stata proprio utilizzata. Paradossalmente, un'ipotetica auto usata solo una volta in tanti anni (un giorno solo!) otterrebbe un valore medio di utilizzi giornalieri = 1 .... che non è proprio il risultato desiderato.*

*(b) Una formulazione possibile calcola la media basandosi sull'attività giornaliera:*

```
create view DAILYACTIVITY( PIno, Day, NumUs, TotTime ) as  
  select PlateNo, Date, count(*), sum(EndTime –StartTime)  
  from USAGE  
 group by PlateNo, Date
```

```
select PlateNo, avg(NumUs) as AvgDailyUses, sum(TotTime)/sum(NumUs)  
as AvgUseDuration  
from DAILYACTIVITY  
group by PlateNo
```

(c) Comunque, nella versione (b), i giorni in cui una data auto non è stata usata non sono considerati quando si fa la media della durata per quell'auto. Un modo "semplice" di contare tutti i giorni è il seguente:

```
create view DAYSOFOPENING( NumDays ) as
```

```
  select count( distinct Date )   assuming that no working day ends with zero cars being used overall  
  from USAGE
```

```
select PlateNo, count(*)/NumDays as AvgNumOfDailyUses, avg(EndTime –StartTime)  
as AvgUseDuration
```

```
from USAGE, DAYSOFOPENING   (not a join, but a "Cartesian product" with just one number)  
group by PlateNo
```

*Dove la view estrae semplicemente un numero (il numero di giorni "validi", che è ovviamente indipendente dall'auto specifica).*



*(d) La view in (c) restituisce sicuramente solo un valore, quindi può essere direttamente inclusa nella lista target della Select, rendendo la definizione della view ridondante. Poi, possiamo anche aggiungere le auto che possibilmente non sono mai state usate (per qualsiasi ragione):*

```
select PlateNo,  
       count( all StartTime ) / ( select count( distinct Date )  
                                   from USAGE ) as AvgNumOfDailyUses,  
       avg(EndTime –StartTime) as AvgUseDuration  
from USAGE U right join CAR C on U.PlateNo = C.PlateNo  
group by C.PlateNo
```

*Si noti che in questa formulazione:*

- (i) il criterio di raggruppamento è C.PlateNo (si sceglie il PlateNo che non è mai nullo, cioè quello di Car, dato che invece U.PlateNo potrebbe essere eventualmente nullo);*
- (ii) Il count non è più count(\*), perché conterebbe 1 anche per le auto che non sono mai state usate, ma è il count di un attributo della tabella USAGE (un altro attributo, ad esempio BadgeNo, andrebbe bene lo stesso).*