Assertion

week 3

- General constraints on dB.
- Create Assertion <name>
  check (<condition>);

ex:

```
C. A. myAssert Check
(NOT EXISTS (
    SELECT P.name
    FROM P, Pr
    where P.name = Pr.name
    group by P.name
    having count(*)>200))
```

→ no item should be sold more than 200 times

Wherever a modification happens (deletion, insertion, update), assertions will be checked. Not practical.
on the table/relation

ex:

```
C. A. Fewbar check(
    (select count(*) from bars) <=
    (    //     from drinkers))
```

→ There cannot be more bars than drinkers.

For better performance, deletion on bars and insertion on drinkers should not invoke this assertion. Therefore, use triggers.

Trigger

event → ins, up, del. to the relation

condition → can be expressed as either before or after the event

if condition true

action

row trigger

- exe only for only modified tuple

ex: When price is ↓, set category 'on sale'

```
C. T. Productcategories
after update of price on Product
referencing
        new row as newtuple
        old row as oldtuple
for each row
when ( Old.price > new.price) ]
```

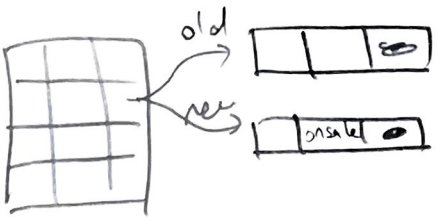
event

condition

```
Update Product
set cat = 'on sale'
where pid = oldtuple.pid
```

action

- DB will hold the table in the ram.
- When price is changed it knows
- It keeps copy of old and new row before storing it permenent



Statement trger

```
after update
referencing
        new table as ---
        old table ------

begin
    update product
    ------
    ------

end
```

Instead of F.K., use triggers to insert a beer without manuf.

Sells (bar, beer, price) ⟶ if there is FK beer, then I cannot insert beer1 if there is no entry in Beer table
Beer (beer, manuf)

C. T. Beertrig
after insert on Sells
ref. new row as newtuple
for each row
when (new.beer not in (
select name from beers))
insert into beers(name)
values (new.beer);

---

insert into sells values (---) ⟶ 1 insert
insert into sells select * from ---- ⟶ Bulk insertion (more than 1)
↳ that's why we use ⟶ referencing new row as newrow
for each row

-Drop Product → delete table from dB
-delete Product → delete rows in product table
-drop Purchaseprice → we can drop a trigger
-alter Product add column ---
-create or replace trigger ... ⟶ if trigger exists, update
else create

---

whether you have after or before in trigger, the modification will not be stored until trigger finishes.

after update of price on Sells → if we say after update on sells, it is not very efficient

---

- You cannot write ~~trigger~~ inside another trigger. But you can design a dB where trigger causes a sequence of rules. Dangerous.

---

after delete
- old row ✓
- ~~new row~~

after insert
- ~~old row~~
- new row ✓

- Empty tables will evaluate true. Empty table does not violate assertion or ~~trigger~~

Views

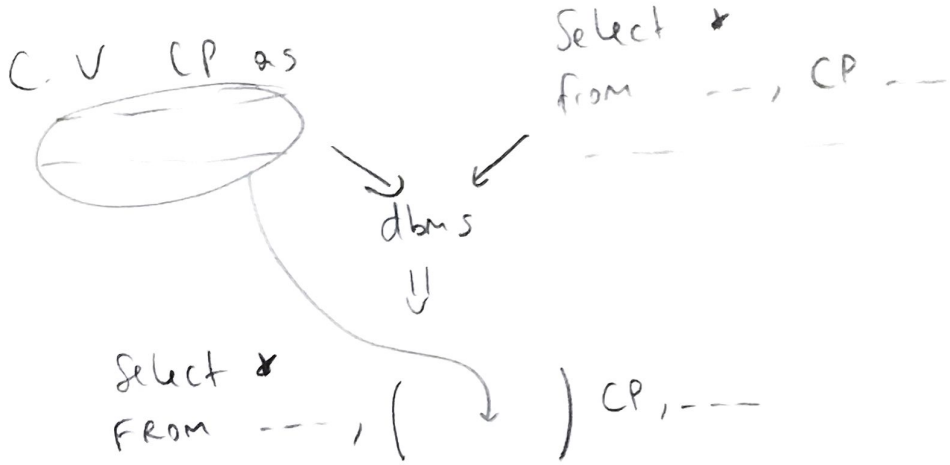
- For representing different info to diff users.
- It physically does not exist ☆
- C. Views CP as → virtual table

→ Virtual view
- Computed on demand, slow at runtime
- always up to date

Materialized View
- Pre computed, fast on runtime
- May have stale data
- In big data warehouses, it (materiel) is stored in the harddrive (physically)

Create [Materialized] View
↓
optional
default is virtual.

- DBMS will unnest the view into query



ex: C.V. Cust.Pr. as
S x.customer, y.price
F Pur x, Pro y
W x.pro = y.pname

⇒ ①

S u.customer, v.store
F CustPr u, Purc v
W u.cust = v.cust and
u.price > 100

① ⇒

S u.cust, v.store
F. (S x.cust, y.price
F Pur x, Pro y
W x.pro = y.pname) u, Purc v
W u.cust = v.cust and
u.price > 100

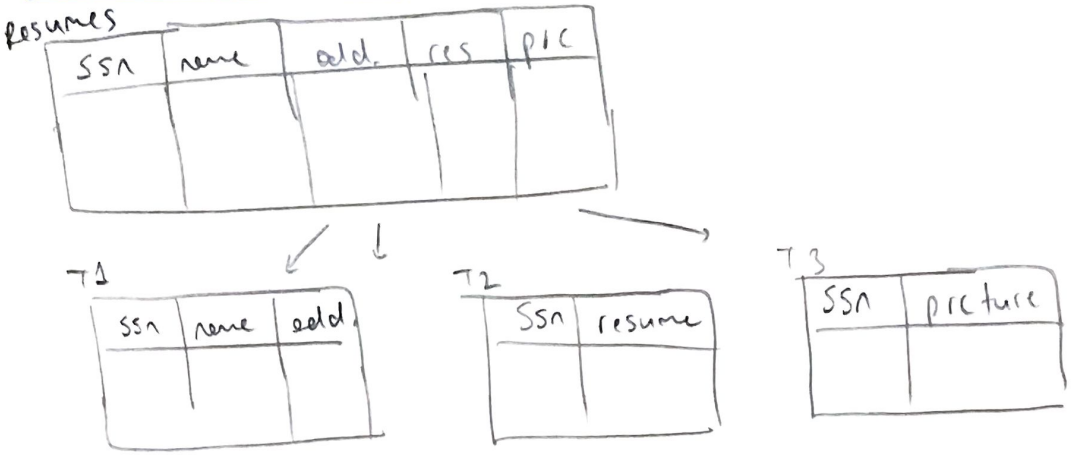
⇓ ② modification

S x.cust, v.store
F Pur x, Pro y, Pur v
W x.cust = v.cust and
y.price > 100 and
x.product = y.pname

Applications

- Physical data independence
  - ↳ vertical / horizontal part
- Security (who can see which table)

Vertical Partitioning




```
C.V  Resumes AS
Sel  T1.SSN, T1.name ---
From T1, T2, T3
W    T.ssn = T2 SSN and T2---
```

→ (assume we designed db s.t. we have T1, T2 and T3. Now we want to combine them for a view)

The query → ☆

```
S  add
F  Resumes
W  name = 'Sue'
```

→

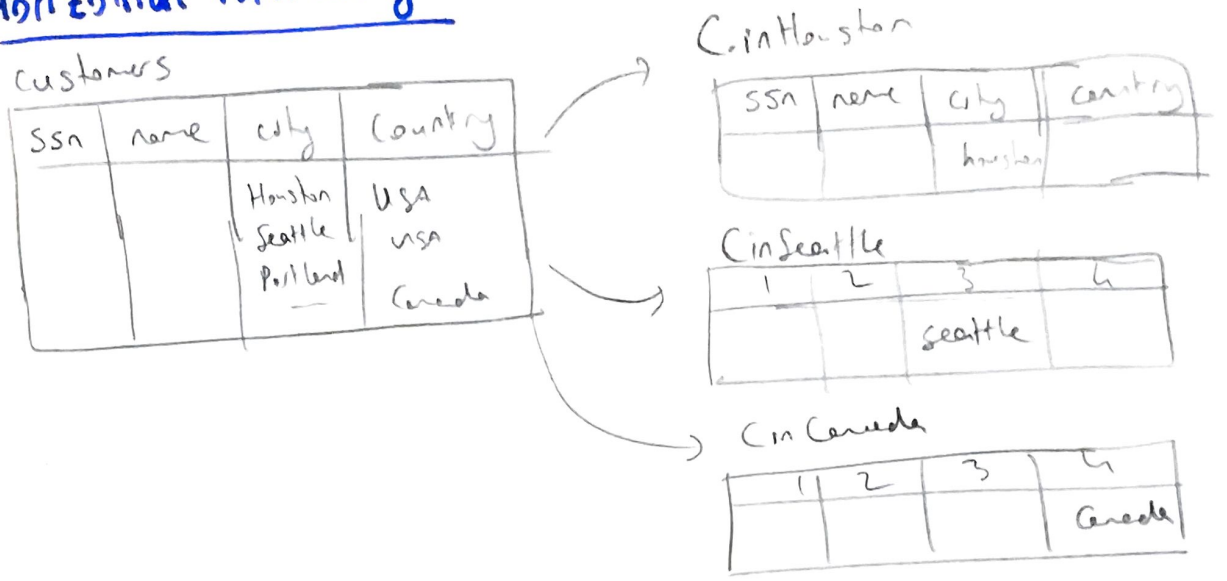
```
S  add
F  ( S  ssn, name, add. --
     F  T1, T2, T3
     W  T1.ssn = T2 ---  )
W  name = 'Sue'
```

If the dbms is clever, it will optimize the query above. But, not all dbms's are clever

When to do vertical part?

- Some fields are ~~large~~ and rarely accessed → picture
- distributed db
- T1 comes from 1 source, T2 comes from another

Horizontal Partitioning



Assume that we have CinHouston, Cin Seattle ---
and we want to combine them.

C. V. Customers as
CinHou
UNION all
CinSea

→ if we say

Sel name
F Customers
W city = 'Seattle'

dbms cannot optimize this.
It will bring all data then eliminate non seattle. Too much work.

However, if we create our view in this format

C V Custom
(Sel. *
F CH W city = 'Seattle')
union all
(S * F CH W city = 'Houston')

→ Compiler can optimize the query above Before doing execution.

Views And Security

C

name	addr	balance

→ Some users are not allowed to see this column

C.V PublicC.
S name, addd
F Customers → ✓

C

name	addr	balance
		+450
		+20
		-30

→ Some are not allowed to see >0 balance

C.V --
S *
F Cust W Balance<0

Privileges

Select, Insert, Delete, Update
↳ row based
↳ may apply to only one attribute

Grant <List of privileges>
ON <relation or objects>
TO <db user>

→

Grant Sel, Upd (address)
ON PubCust
TO Joe

Joe can select and update the addres of PubCust

Updating Views

-A view update changes underlying table → produce requested change to the view.

Transkript (Studid, Crscode, Semester, Grade)

C.V. CsgReg (Studid, crscode, semester) as
S T.studid, T.crscode, T.semester
F Tran. T
W T.crscode Like 'Csg%' and T.Semester='S2013'

Q: insert into CsgReg (---) vals(1111, Csg352, S2013)
what will be the grade?

A: In transcript table grade will be NULL (if it is allowed)

Q: insert into CsgReg (--) vals (1111, Econ210, S2013)
New tuple will not be in the view.

A: Allow insertion WITH CHECK OPTION clause on the Create View

---

Professor (Id, name, dept)
Dept (deptId, name)

C.V. ProfDept (Prname, Dename) as
S P.name, D.name
F Pr P, De D
W P.deptid = D.id

Q: delete <Smith, Csg> From ProfDept

Ambiguity

1) Should professor be deleted??
2) Should Csg dept be deleted??
3) Update row for smith deptid=null??
↳ Good, but computer cannot know this.

- We cannot write Foreign Keys in Views for solving ambiguity.
- But we can write Triggers!

Some Views are not Updatable

```
C.V  AvgSalary (deptid, avgsal) as
S    E.deptId, AVG (E.salary)
F    Emp E
Group By E.deptId
```

How do we handle (update AvgSalary set AvgS = 1.1 * Avgsal) ?

Employee (ssn, name, dept, project, salary)

```
C.V. Developers as
S    name, project
F    Emp
W    dept = 'Development'
```

→ insert into developers vals (Joe, 'optimizer')

⇓

insert into Employee vals

(null, Joe, null, optimizer, null)

- null: assume it is allowed
- null: This must be development

To solve the issue above, we can use triggers.

```
C.T  AllowInsert
Instead OF insert on Developers
Referencing new row as newrow
For each row
Begin
    insert into Emp (name, dept, proj)
    Vals (newrow.name, 'Development', new.project)
End
```

Materialized Views

- Each time base table change, matview may change.
- We cannot afford to recompute the view with each change.
- Soln: Periodically construct matview.

Indexes

- Hash table or B+tree (default)

Syntax →
```
Create Index CatIndex on Product(cat)
C   I   EmpInd on Emp(name) USING HASH;
_   _   _  Transcript(semester,course) USING BTREE;
```
(semester: 1st semester; course: 2nd course)

Q: Find prices of beers manuf by Tuborg and sold at Joe's bar

```
C. I.  BeerInd on Beers(manuf)
C_I.   SellInd on Sells(bar,beer)
```

Query
```
S  price
F  Beers, Sells
WH  manf = Tuborg and
    B.name = S.beer and
    bar = Joe's bar
```
→ much faster.

✡ If insert, delete, update on the index is very frequent, these operation → will be slow.