Join Operation:

Takes the Cartesian Product of a Two Tables while repeating The join Constraint

* Invalid Rows: the ones that violate the join Constraint

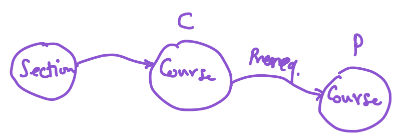
* Advantage: More Efficient Compared to Cartesian Product, bcz the invalid rows don't get Generated

```
Select [Attributes]
From
    [T1] JOIN [T2] ON [C1]
         JOIN [T3] ON [C2]
         ...
         JOIN [Tk] ON [Ck-1]
where
         [Additional Constraints]
group by ~
order by --
```

Name & ID of 'CS' Professors

```
Select Professor.name, Professor.ID
from
      Professor JOIN department
       ON Professor.DepID = Department.ID
where
   Department.name = 'CS'
```

- The Prerequisites of Section 'CS480S1'



```
Select P.ID
from
   Section join Course as C
                 on Section.CourseID = C.ID
      join Course as P
                on C.PrereqID = P.ID
      where Section.ID = 'CS480S1'
```

```
Select Course.PrereqID
from
     Section Join Course
        On Section.CourseID = Course.ID
       Where Section.ID = 'CS480S1'
```

- The ID of Courses a Student has taken

```
Select Section.CourseID
from
     Register join Section
        on Register.SectionID = Section.ID
     where
          StudentID = '...'
```

```
Select  Student.Name , Student.ID
from  Student   join  Register as R
      On  R.StudentID = Student.ID
      where
      R.SectionID = 'CS480S1'
      AND
      NOT EXISTS (( Select  Course.PrereqID
                    from
                       Section  join  Course
                          on  Section.CourseID = Course.ID
                          Where  Section.ID = 'CS480S1'
                  )
                  EXCEPT

                ( Select  Section.CourseID
                  from
                     Register  join  Section
                        on  Register.SectionID = Section.ID
                     where

                     Register.StudentID = Student.ID   ))
```

Natural join: Sets the join Condition on the Columns with the Same Names.

e.g.,

T_1

C_1	C_2
A	1
B	3
C	1
D	2
E	5

T_2

C_2	C_3
4	α
2	β
3	γ
2	δ
1	σ

Inner Join

T_1 Natural Join T_2

C_1	C_2	C_3
A	1	σ
B	3	γ
C	1	σ
D	2	β
D	2	δ

Select *
from T1 Natural join T_2

---

e.g., Select *
from Professor Natural join Department

→ Always returns [Empty Set]

↳Reason: Columns NAME in both Department and Professor Tables

Join
- Inner Join (Default): Only includes rows that match both sides
- Outer Join: Includes the rows that **do not** have a match in the other Table

Outer Join
- Left outer join: Includes the unmatched rows of the left Table (T_1 Left outer join T_2)
- right " " : " " " " " " Right Table (T_1 Right outer join T_2)
- Full outer join: " " " " " " of either Side (T_1 outer join T_2)

e.g.

T_1 left outer join T_2

C_1	C_2	C_3
A	1	σ
B	3	γ
C	1	σ
D	2	β
D	2	δ
E	5	Null

T_1 Right outer join T_2

C_1	C_2	C_3
A	1	σ
B	3	γ
C	1	σ
D	2	β
D	2	δ
Null	4	α

T_1 outer Join T_2

C_1	C_2	C_3
A	1	σ
B	3	γ
C	1	σ
D	2	β
D	2	δ
Null	4	α
E	5	Null

VIEWS : A logical Table that is derived from the existing Tables

– Create Views :

CREATE VIEW [View_name] AS
[Select query]

e.g., Create a view of Departments & the Count, Sum, min, max, Avg Salary

```
Create View DepStats (ID, Cnt, Sum, max,
                      min, Average) AS
   Select DepID, Count(*), Sum(Salary),
          max(Salary), Min(Salary), Avg(Salary)
   from Professor
   Group by DepI
```

– Find the Average Salary of the 'CS' Department.

```
Select Average
from Department join DepStat on
                     Department.ID = DepID
where
   Name = 'CS'
```

– To Create Materialized Views (physically Stored, Pre Computed)

– CREATE MATERIALIZED View ...

Updating [Insert / Delete / Update] views

e.g.,

```
Create View Prof_Public AS
  Select UIN, Name, DepID
  from Professor
```

– Delete Prof_Public
 where
   UIN = '...'   ✓

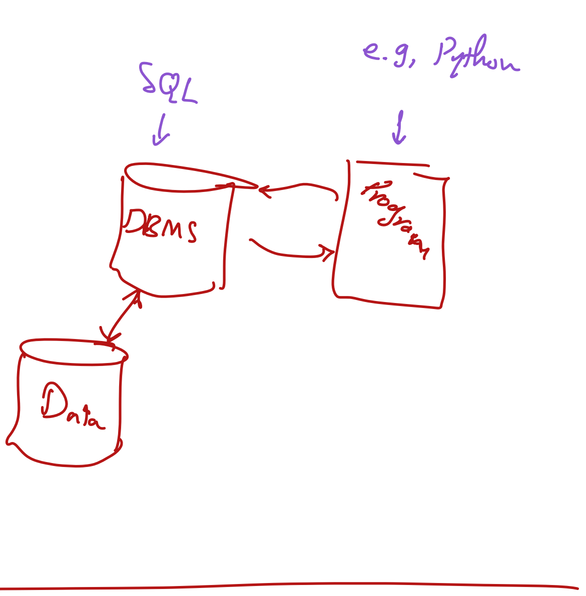
↳ deletes a row from the Table PROFESSOR

– Insert into Prof_Public
   values ('—', A, 1)

✓: only if unspecified Columns Can be Null or Auto-Gen.

Transaction: A set of operations (queries) that either all or none of them should get executed

```
Start Transaction
    ...
    ...
Rollback; // None of the updates should get reflected
Commit; // The Transaction operations are complete
```



eg. Transfer $100 from SrcID to DesID

```
St = " Start Transaction "
St += "         Update Account "
St += "         Set balance = balance - " + toStr(Amount);
                where ID = SrcID
                Update Account
                Set balance = balance + " + toStr(Amount);
                where ID = Des ID
St+= "Commit; "
DB.query(St)
```

```
Create   Procedure   Transfer(
                              IN  SrcID   INT,
    Start  Transaction        IN  DesID   INT,
                              IN  Amount  Decimal(20,2))
        Declare  B  Decimal(20,2)
        Select  Balance  INTO  B
        from   Account
        where  ID = SrcID
        If  B < Amount  Then
              Rollback
        Update  Account
        Set    balance = balance - Amount
        where  ID = SrcID

        Update  Account
         Set    balance = balance + Amount
         where  ID  = Des ID
    Commit;
  End;
```

- Execute a Procedure

```
Call  Transfer(_, _, _)
```