**Linux:-**

Find and replace file extension

-------------------

find . -type f -name "\*.pck" -exec rename 's/\.pck$/.sql/' {} +

-----------------

find . -type f -name "\*.pck" | while read file; dos

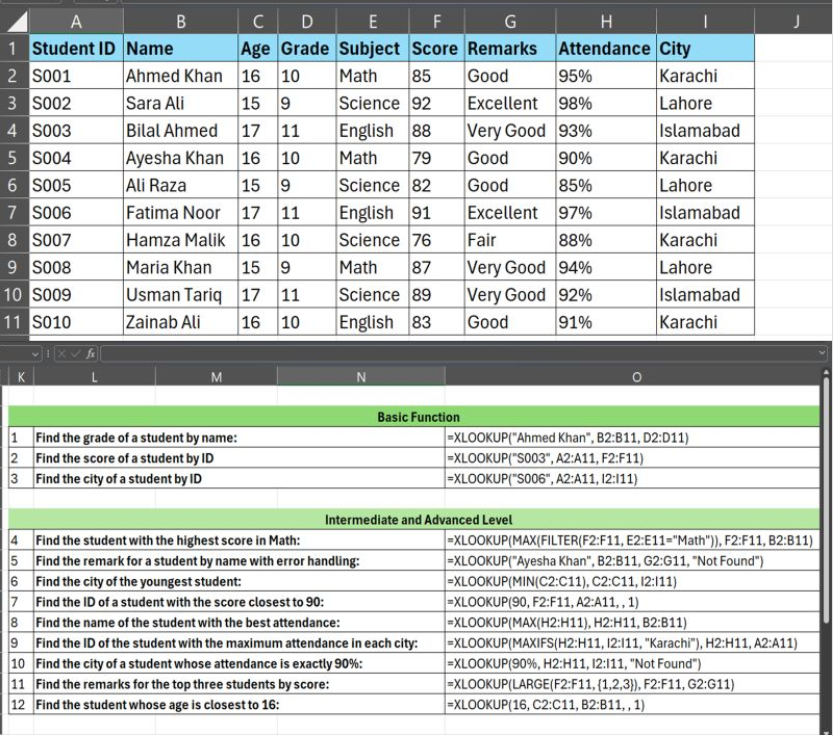
mv -- "$file" "${file%.pck}.sql"

done

**TechLake:-** Complex Data types <https://www.youtube.com/watch?v=qfJb45SusMo&list=PL50mYnndduIF868zbDUPMBpJpwJwd4NZh&index=3>

𝐑𝐨𝐮𝐧𝐝𝟏- 𝐂𝐨𝐝𝐢𝐧𝐠 𝐑𝐨𝐮𝐧𝐝 [ 𝐉𝐢𝐫𝐚 𝐏𝐥𝐚𝐭𝐟𝐨𝐫𝐦 𝐂𝐫𝐞𝐚𝐭𝐢𝐨𝐧 𝐛𝐲 𝐏𝐲𝐭𝐡𝐨𝐧 𝐎𝐎𝐏𝐒 ]  
  
User takes subscription of Jira as premium or standard  
User should get the total cost of the subscription at end of the month  
If he subscribes in middle of month he should get prorated data  
  
I started the coding in Python and used the Python OOPS concepts to create various kinds of classes and methods.  
  
After coding upto 70% of code the interviewer asked me to explain and gave me few cross questions.  
  
𝐑𝐨𝐮𝐧𝐝𝟐 - 𝐒𝐑𝐄 𝐜𝐫𝐚𝐟𝐭 𝐢𝐧𝐭𝐞𝐫𝐯𝐢𝐞𝐰  
  
The interviewer gave me various kinds of elements and asked me to implement the best SRE/Observability practices  
  
Client -> Load balancer -> Web tier -> DB tier -> Messaging queue -> SMTP -> Process -> and so on  
  
This round went for 1 hour and the discussion was all about the SRE techniques.  
  
𝐑𝐨𝐮𝐧𝐝𝟑 - 𝐕𝐚𝐥𝐮𝐞𝐬 𝐑𝐨𝐮𝐧𝐝  
  
In this round the questions were around the following topics  
  
Be the change you seek  
Play, as a team  
Build with heart and balance  
Don’t #@!% the customer  
  
𝐑𝐨𝐮𝐧𝐝 𝟒 - 𝐌𝐚𝐧𝐚𝐠𝐞𝐫𝐢𝐚𝐥/𝐓𝐞𝐜𝐡𝐧𝐢𝐜𝐚𝐥  
  
I was expecting only managerial questions but in this round it was technical 60%+ questions based on DOCKER, JAVA, INFRASTRUCTURE, TERRAFORM etc.

**Excel:-**



**SQL :-**

------ ILT staff related information -----  
  
**with** **recursive** *numbers*(n) **as** (  
**select**  
1 **as** *n*  
**union** **all**  
**select**  
***n*** + 1  
**from**  
numbers  
**where**  
***n*** <= 1000  
-- Define the termination condition here  
),  
*staff\_info* **as** (  
**select**  
*ilts*.***learn\_training\_id*** **as** *ilt\_learn\_training*,  
*ilts*.***version*** **as** *version*,  
*ilts*.***staff\_type*** **as** *ilt\_staff\_type*,  
**split\_part**(*ilts*.***staff\_user\_ids***,  
**','**,  
*numbers*.*n*) **as** *ilt\_staff\_user\_id*,  
**split\_part**(*ilts*.***staff\_registration\_status***,  
**','**,  
*numbers*.*n*) **as** *ilt\_staff\_registration\_status*  
**from**  
***andes***.***learn***.***ilt\_staff*** *ilts*  
**join** *numbers* **on**  
**split\_part**(*ilts*.***staff\_user\_ids***,  
**','**,  
*numbers*.*n*) <> **''**  
--where  
--ilts.learn\_training\_id = 'TPTLERN20240702220824fd937171'  
)  
**select**  
*si*.*ilt\_learn\_training* **as** *ilt\_training\_id*,  
*si*.*ilt\_staff\_type* **as** *ilt\_staff\_type*,  
*u*.***id*** **as** *ilt\_user\_id*,  
*u*.***papi\_displayname*** **as** *ilt\_staff\_display*,  
*u*.***papi\_bi\_jobstatus*** **as** *ilt\_staff\_jobstatus*,  
*u*.***papi\_bi\_login*** **as** *ilt\_staff\_bi\_login*  
**from**  
*staff\_info* *si*  
**left** **join** ***andes***.***learn***.***users*** *u* **on**  
*si*.*ilt\_staff\_user\_id* = *u*.***id***;

**Path and courses:-**

**select**   
*cls*.***lp\_learn\_training\_id*** **as** *lp\_id*,  
*cls*.***lo\_learn\_training\_id*** **as** *learn\_course\_id*,   
*tt*.***training\_title*** **as** *LP\_training\_title*,  
*t*.***learn\_training\_id*** **as** *training\_id*,  
*ttl*.***training\_title*** **as** *co\_training\_title*,  
*t*.***deep\_link*** **as** *Learning\_path\_deep\_link*,  
*t*.***training\_type*** **as** *training\_type*,   
*t*.***learning\_platform\_name*** **as** *training\_platform\_name*,   
*t*.***training\_status*** **as** *training\_status*,  
*cls*.***lp\_version*** **as** *co\_lp\_version*,   
*t*.***version*** **as** *lo\_course\_version*,  
cls.**index** **as** *course\_sequence*  
**from**   
***andes***.***learn***.***course\_lp\_sequence*** *cls* ,   
***andes***.***learn***.***training*** *t* ,   
***andes***.***learn***.***training\_title*** *tt*, -- Training\_Title for LP course  
***andes***.***learn***.***training\_title*** *ttl* -- Training\_Title for courses  
**where**   
*cls*.***lp\_learn\_training\_id*** = **'LPRLERN202406172100391d036df9'** **and**   
*cls*.***lo\_learn\_training\_id*** = *t*.***learn\_training\_id*** **and**  
*cls*.***lp\_learn\_training\_id*** = *tt*.***learn\_training\_id*** **and**  
*cls*.***lo\_learn\_training\_id*** = *ttl*.***learn\_training\_id*** **and**  
***lp\_version*** = (  
**select** **max**(***lp\_version***) **from**   
***andes***.***learn***.***course\_lp\_sequence*** *cls1*  
**where** *cls*.***lp\_learn\_training\_id*** = *cls1*.***lp\_learn\_training\_id***  
**and** *cls*.***lp\_version*** **not** **like** **'%.%.%'**  
) **and**   
*t*.***version*** = (**select** **max**(*t1*.***version***) **from** ***andes***.***learn***.***training*** *t1*   
**where** *cls*.***lo\_learn\_training\_id*** = *t1*.***learn\_training\_id***  
) **and**   
*ttl*.***version*** = (**select** **max**(***version***) **from**   
***andes***.***learn***.***training\_title*** *title* **where** --title.version not like '%.%.%' and  
*title*.***learn\_training\_id*** = *ttl*.***learn\_training\_id***) **and**  
*tt*.***version*** = (**select** **max**(***version***) **from**   
***andes***.***learn***.***training\_title*** *tt1* **where** *tt*.***learn\_training\_id*** = *tt1*.***learn\_training\_id*** **and**   
*tt1*.***version*** **not** **like** **'%.%.%'**  
);

**Example of split part:-**

WITH cte AS (  
SELECT   
SPLIT\_PART(SPLIT\_PART(t1.id, '#', 4), ':', 2) AS split\_value,   
t1.\*,  
t1.trainingId,  
u.lme -- Ensure 'lme' is selected from the 'users' table  
FROM   
andes.learn.transcripts t1  
JOIN   
andes.learn.users u ON t1.userid = u.id  
JOIN   
andes.learn.training\_title ttl ON ttl.learn\_training\_id = t1.trainingId  
JOIN   
andes.learn.training tr ON tr.learn\_training\_id = t1.trainingId  
WHERE   
u.papi\_bi\_login IN ('shaikafq')  
AND ttl.version = tr.version  
AND ttl.version = (  
SELECT MAX(version)   
FROM andes.learn.training\_title t2   
WHERE t2.learn\_training\_id = ttl.learn\_training\_id   
AND t2.version NOT LIKE '%.%.%'  
)  
AND tr.version = (  
SELECT MAX(version)   
FROM andes.learn.training\_title t2   
WHERE t2.learn\_training\_id = tr.learn\_training\_id   
AND t2.version NOT LIKE '%.%.%'  
)  
)  
SELECT \*  
FROM cte  
WHERE (  
**u.lme = 'KNet' – Execute below condition only if lms = ‘kNet’**  
AND split\_value = (  
SELECT MAX(split\_value)  
FROM cte AS inner\_cte  
WHERE inner\_cte.trainingId = cte.trainingId  
)  
)  
**OR u.lme <> 'KNet'; -- If not run select \* on CTE**

**Example of NEJ:-**

with section\_range as (

select ac.curriculum\_id , ac.parent\_training\_id , ac.training\_id , att.training\_title as training\_title , ac.training\_display\_sequence ,

LEAD(training\_display\_sequence , 1) OVER(

ORDER BY training\_display\_sequence

) AS next\_Section\_id,

lag(training\_display\_sequence , 1) OVER(

ORDER BY training\_display\_sequence

) AS prev\_section\_id,

ac.al\_lms

from

"knet-data".alrn\_curriculum ac,

"knet-data".alrn\_training\_title att

where --ac.curriculum\_id in ('028a2e92-108f-4390-84c4-3e1d27446c5e') and

ac.curriculum\_id in ('028a2e92-108f-4390-84c4-3e1d27446c5e') and

ac.required\_training\_per\_section is not null and --for section\_title

att.training\_id = ac.training\_id and

att.is\_default\_language = 'True' and

att.al\_lms = ac.al\_lms

order by training\_display\_sequence

),

curr as(

select ac.curriculum\_id , ac.parent\_training\_id , ac.training\_id , att.training\_title as training\_title\_t , ac.training\_display\_sequence

, lag(training\_display\_sequence , 1) OVER(

ORDER BY training\_display\_sequence

) AS previous\_section\_id ,att.training\_title ,

case when ac.required\_training\_per\_section is null then 'Training' else 'Section\_Title' end as curriculum\_section\_part ,

ac.al\_lms

from

"knet-data".alrn\_curriculum ac,

"knet-data".alrn\_training\_title att

where ac.curriculum\_id in ('028a2e92-108f-4390-84c4-3e1d27446c5e') and

--ac.required\_training\_per\_section is null and

att.training\_id = ac.training\_id and

att.al\_lms = ac.al\_lms and

att.is\_default\_language = 'True'

order by training\_display\_sequence

)

select

sr.curriculum\_id,

att.training\_title as curriculum\_title,

c.parent\_training\_id,

sr.training\_title as section\_title,

c.training\_id as section\_training\_id,

at2.training\_type\_description ,

c.training\_title\_t as trainings\_under\_section

from

section\_range sr,

curr c ,

"knet-data".alrn\_training at2 ,

"knet-data".alrn\_training\_title att

where (

(c.training\_display\_sequence between sr.training\_display\_sequence and sr.next\_Section\_id )

or (sr.next\_section\_id is null and c.training\_display\_sequence > sr.training\_display\_sequence)

or (sr.prev\_section\_id is null and sr.training\_display\_sequence <> 1 and c.training\_display\_sequence < sr.training\_display\_sequence)

)

and

c.curriculum\_section\_part <> 'Section\_Title' and

c.training\_id = at2.training\_id and

c.al\_lms = at2.al\_lms and

att.training\_id = sr.curriculum\_id and

att.al\_lms = sr.al\_lms

order by c.training\_display\_sequence asc, sr.training\_display\_sequence asc;

**Merge Statements:-**

MERGE INTO target\_table t

USING source\_table s

ON (t.id = s.id)

WHEN MATCHED THEN

    UPDATE SET

        t.column1 = s.column1,

        t.column2 = s.column2

WHEN NOT MATCHED THEN

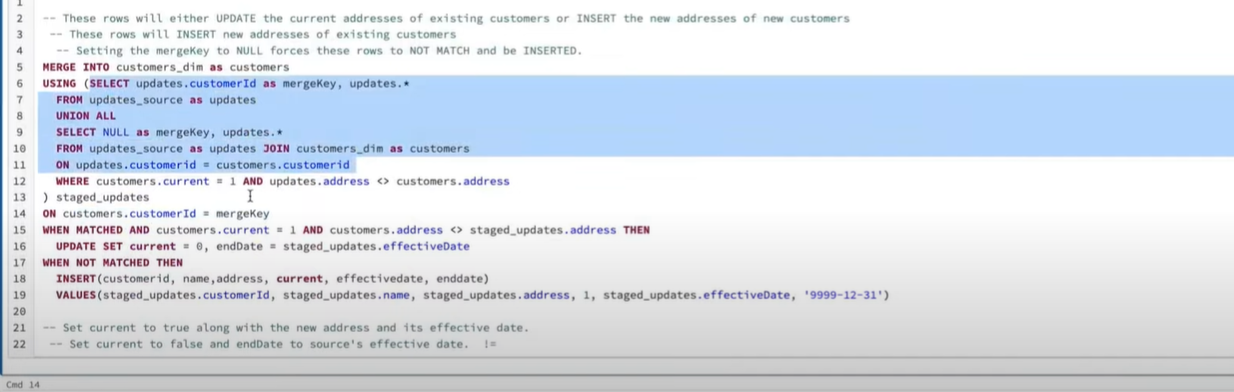
    INSERT (id, column1, column2)

    VALUES (s.id, s.column1, s.column2);

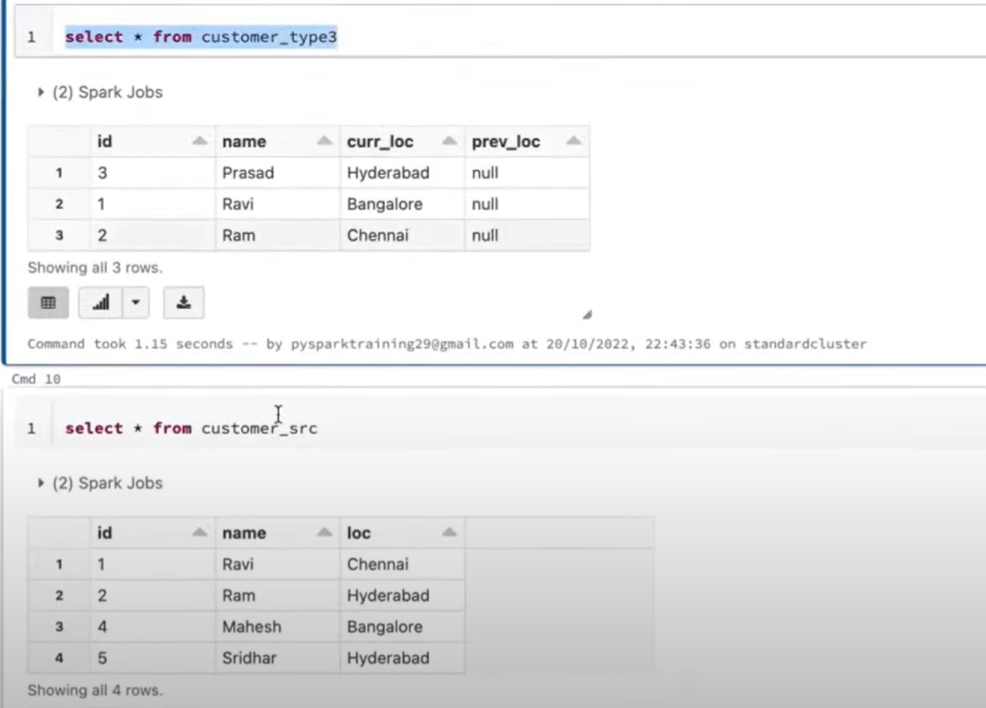
**TechLake:-** Complex Data types

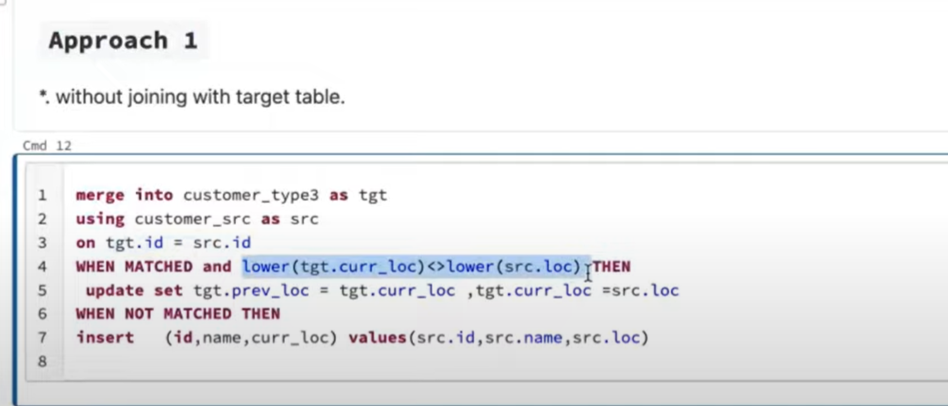
<https://www.youtube.com/watch?v=qfJb45SusMo&list=PL50mYnndduIF868zbDUPMBpJpwJwd4NZh&index=3>

**SCD Type 2 Using merge:-**



**SCD Type 3:- Additional Column case**







**What is Normalization in SQL?**

Normalization is used to decompose a larger, complex table into simple and smaller ones. This helps us in removing all the redundant data.

Generally, in a table, we will have a lot of redundant information which is not required, so it is better to divide this complex table into multiple smaller tables which contain only unique information.

**First normal form:**

A relation schema is in 1NF, if and only if:

* All attributes in the relation are atomic(indivisible value)
* And there are no repeating elements or groups of elements.

**Second normal form:**

A relation is said to be in 2NF, if and only if:

* It is in 1st Normal Form.
* No partial dependency exists between non-key attributes and key attributes.

**Third Normal form:**

A relation R is said to be in 3NF if and only if:

* It is in 2NF.
* No transitive dependency exists between non-key attributes and key attributes through another non-key attribute

Example of

Normalization in SQL involves organizing data to reduce redundancy and improve data integrity. Let's break down the three normal forms (1NF, 2NF, 3NF) with an example.

### Problem Scenario

Assume we have a table Student\_Courses:

| **StudentID** | **StudentName** | **CourseID** | **CourseName** | **Instructor** | **InstructorOffice** |
| --- | --- | --- | --- | --- | --- |
| 101 | John Doe | C101 | Mathematics | Dr. Smith | Room 12 |
| 102 | Jane Doe | C102 | Physics | Dr. Brown | Room 15 |
| 101 | John Doe | C102 | Physics | Dr. Brown | Room 15 |

This table contains data about students, courses, and instructors. However, it's not normalized yet.

### **1st Normal Form (1NF) - Eliminate Repeating Groups**

* **Rule**: Each column must contain atomic (indivisible) values, and each record must be unique.
* **Issue**: The table already follows 1NF because it has atomic values. However, there are duplicate rows, and StudentName, CourseName, Instructor, and InstructorOffice appear multiple times for the same student or instructor.

So, we identify the primary key: (StudentID, CourseID).

| **StudentID** | **StudentName** | **CourseID** | **CourseName** | **Instructor** | **InstructorOffice** |
| --- | --- | --- | --- | --- | --- |
| 101 | John Doe | C101 | Mathematics | Dr. Smith | Room 12 |
| 102 | Jane Doe | C102 | Physics | Dr. Brown | Room 15 |
| 101 | John Doe | C102 | Physics | Dr. Brown | Room 15 |
|  |  |  |  |  |  |

### **2nd Normal Form (2NF) - Eliminate Partial Dependency**

* **Rule**: 1NF + no partial dependency (non-key columns must depend on the whole primary key).
* **Issue**: In this table, StudentName depends only on StudentID (not on the whole key (StudentID, CourseID)), and Instructor and InstructorOffice depend only on CourseID.

To resolve this, we split the table into two smaller tables:

1. Student table (depends only on StudentID):

| **StudentID** | **StudentName** |
| --- | --- |
| 101 | John Doe |
| 102 | Jane Doe |

1. Course\_Instructor table (depends only on CourseID):

| **CourseID** | **CourseName** | **Instructor** | **InstructorOffice** |
| --- | --- | --- | --- |
| C101 | Mathematics | Dr. Smith | Room 12 |
| C102 | Physics | Dr. Brown | Room 15 |

1. Student\_Course table (connects StudentID and CourseID):

| **StudentID** | **CourseID** |
| --- | --- |
| 101 | C101 |
| 102 | C102 |
| 101 | C102 |

### **3rd Normal Form (3NF) - Eliminate Transitive Dependency**

* **Rule**: 2NF + no transitive dependency (non-key columns must depend only on the primary key, and nothing else).
* **Issue**: In the Course\_Instructor table, InstructorOffice depends on Instructor, not on CourseID. Therefore, we need another table to store instructor data separately.

1. Instructor table:

| **Instructor** | **InstructorOffice** |
| --- | --- |
| Dr. Smith | Room 12 |
| Dr. Brown | Room 15 |

1. Course table:

| **CourseID** | **CourseName** | **Instructor** |
| --- | --- | --- |
| C101 | Mathematics | Dr. Smith |
| C102 | Physics | Dr. Brown |

### Final Normalized Tables

1. **Student**:

| **StudentID** | **StudentName** |
| --- | --- |
| 101 | John Doe |
| 102 | Jane Doe |

1. **Course**:

| **CourseID** | **CourseName** | **Instructor** |
| --- | --- | --- |
| C101 | Mathematics | Dr. Smith |
| C102 | Physics | Dr. Brown |

1. **Instructor**:

| **Instructor** | **InstructorOffice** |
| --- | --- |
| Dr. Smith | Room 12 |
| Dr. Brown | Room 15 |

1. **Student\_Course**:

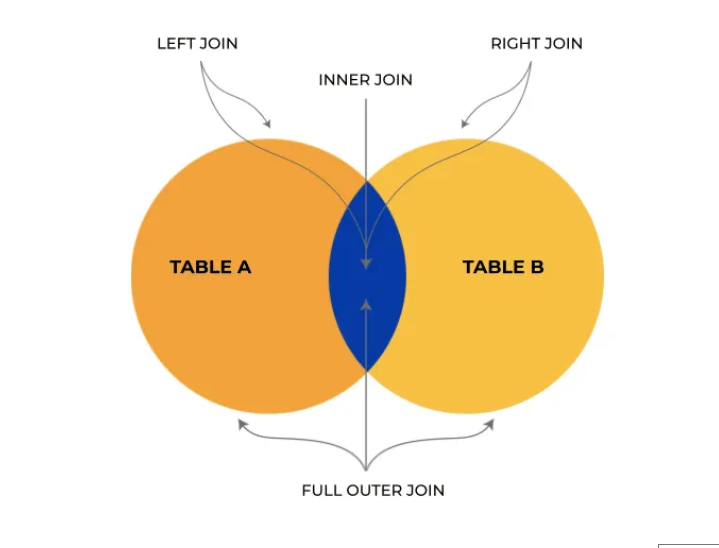
| **StudentID** | **CourseID** |
| --- | --- |
| 101 | C101 |
| 102 | C102 |
| 101 | C102 |

### Explanation:

1. **1NF** ensures atomicity and unique records.
2. **2NF** removes partial dependencies by creating separate tables for entities (Student, Course).
3. **3NF** removes transitive dependencies by separating attributes that don't depend on the primary key (like InstructorOffice).

These steps reduce redundancy and maintain data integrity.

**SQL Join :-**



---------------------------------------------------------------------------------------------

1. NOT IN:

NOT IN checks if a value is not in a list of values.

It can be used with both static lists and subqueries.

Be cautious with NULL values when using NOT IN because if the subquery or list contains NULL, it may cause unexpected behavior.

Example:-

SELECT \*

FROM employees

WHERE department\_id NOT IN (SELECT department\_id FROM closed\_departments);

This query retrieves all employees whose department is not in the list of closed departments.

---------------------------------------------------------------------------------------------

2. NOT EXISTS:

NOT EXISTS is used to check whether a subquery returns any rows. It is generally more efficient than NOT IN, especially when dealing with large datasets.

It doesn’t have the NULL issue like NOT IN.

Example:

SELECT \*

FROM employees e

WHERE NOT EXISTS (SELECT 1 FROM closed\_departments d WHERE e.department\_id = d.department\_id);

This query retrieves all employees whose department does not exist in the closed\_departments table.

---------------------------------------------------------------------------------------------

When to use:

NOT IN: Simpler when comparing a column to a finite list of values.

NOT EXISTS: More suitable when dealing with subqueries, especially when the subquery involves checking multiple conditions or large data.

----------------------------------------------------------------------------------------------

**Recursion/NEJ/CTE**

**with** **recursive** *numbers*(n) **as** (

**select**

1 **as** *n*

**union** **all**

**select**

n + 1

**from**

numbers

**where**

n <= 1000

-- Define the termination condition here

),

*staff\_info* **as** (

**select**

*ilts*.learn\_training\_id **as** *ilt\_learn\_training*,

*ilts*.version **as** *version*,

*ilts*.staff\_type **as** *ilt\_staff\_type*,

split\_part(*ilts*.staff\_user\_ids,

**','**,

*numbers*.*n*) **as** *ilt\_staff\_user\_id*,

split\_part(*ilts*.staff\_registration\_status,

**','**,

*numbers*.*n*) **as** *ilt\_staff\_registration\_status*

**from**

andes.learn.ilt\_staff *ilts*

**join** numbers **on**

split\_part(*ilts*.staff\_user\_ids,

**','**,

*numbers*.*n*) <> **''**

--where

--ilts.learn\_training\_id = 'TPTLERN20240702220824fd937171'

)

**select**

*si*.ilt\_learn\_training **as** *ilt\_training\_id*,

*si*.ilt\_staff\_type **as** *ilt\_staff\_type*,

*u*.id **as** *ilt\_user\_id*,

*u*.papi\_displayname **as** *ilt\_staff\_display*,

*u*.papi\_bi\_jobstatus **as** *ilt\_staff\_jobstatus*,

*u*.papi\_bi\_login **as** *ilt\_staff\_bi\_login*

**from**

staff\_info *si*

**left** **join** andes.learn.users *u* **on**

*si*.ilt\_staff\_user\_id = *u*.id;