**PROBABLE USE CASES OF JOINS IN REAL TIME APPLICATIONS**

1. LEFT JOIN

Left join is used to select all rows from the left table and also the matched values between the two tables.

Here are some probable use cases of left join in real-time applications:

Real-time Marketing Campaign Analysis: Left joins can be utilized in real-time marketing campaign analysis. By performing a left join between a campaign dataset and a customer response dataset, we can include all campaigns in the analysis, even if some customers have not responded yet.

Real-time Website Analytics: Left joins can be applied in real-time website analytics to track user behavior. By performing a left join between a user activity data stream and a user profile data stream, we can include all users in the analysis, regardless of whether they have completed specific actions or have provided additional profile information.

1. RIGHT JOIN

This join returns all the rows of the table on the right side of the join and matching rows for the table on the left side of the join.

Use cases of right join in real-time applications:

Data Migration or Integration: Used in real-time data migration or integration processes when we want to combine data from two sources, ensuring that all records from the right table are included in the result, even if there are no matches in the left table. This can be helpful when merging data from legacy systems or when integrating data from external sources.

Real-time Customer Relationship Management (CRM): Right joins can be applied in real-time CRM systems. For instance, when combining customer data with sales or transactional data, a right join ensures that all customer records are included in the analysis or reporting, even if they have not made any transactions or if some customer records are missing in the primary dataset.

1. INNER JOIN

Inner join selects records that have matching values in both tables as long as the condition is satisfied.

Use cases of inner join in real-time applications:

Real-time E-commerce: Used in real-time e-commerce applications to match customer profiles with product catalogs. By joining customer data with product data, we can provide personalized recommendations, display relevant product information, and offer targeted promotions to customers in real-time.

Real-time Social Media Analysis: Inner joins can be applied to analyze real-time social media data. By joining social media posts with user profiles or sentiment analysis data, we can identify trends, measure sentiment, or track the impact of social media campaigns in real-time.

1. FULL JOIN

It returns all the rows from both joined tables, whether they have a

Matching row or not

Use cases of full join in real-time applications:

Data Synchronization: Full joins can be applied in real-time data synchronization

scenarios. For example, if you have a primary and secondary database that need

to be synchronized, a full join can help identify and reconcile differences between

the two datasets by including all records from both databases, regardless of

matches.

Data Profiling: Full joins can be applied in real-time data profiling and analysis

tasks. By performing a full join between different subsets or segments of a

dataset, you can identify common records, unique records, and discrepancies

between them.

1. CROSS JOIN

The Cross join is used to generate a paired combination of each row of the

first table with each row of the second table. This join type is also known as

Cartesian join.

Use cases of cross join in real-time applications:

Real-time Testing or Quality Assurance: Cross joins can be used in real-time

testing or quality assurance processes to ensure comprehensive coverage of

test cases. By combining various input datasets or configurations using a cross

join, you can test all possible combinations and uncover potential issues or edge

cases in real-time applications.

Real-time Product Catalog Generation: Cross joins can be utilized to generate

product catalogs or combinations of items. For example, in a real-time

e-commerce application, you may want to create all possible combinations of

products for bundling or create cross-sell or upsell recommendations based on

combinations of items.

**NORMALIZATION**

Normalization is the process of organizing the data in the database by

minimizing the redundancy from a relation or set of relations.

TYPES OF NORMAL FORMS

1) First Normal Form (1NF)

* In 1NF, each table should contain a single value.
* Each record needs to be unique.

Example: Consider an Employee table with employees in multiple departments.

Employee

| Employee Name | Age | Department |
| --- | --- | --- |
| Ram | 34 | Marketing ,Sales |
| Sonu | 32 | Quality Assurance |
| Nita | 35 | Human Resource ,Sales |

Employee table after 1NF will be,

Employee

| Employee Name | Age | Department |
| --- | --- | --- |
| Ram | 34 | Marketing |
| Ram | 34 | Sales |
| Sonu | 32 | Quality Assurance |
| Nita | 35 | Human Resource |
| Nita | 35 | Sales |

2) Second Normal Form (2NF)

* The table should be in 1NF
* All non-key attributes are fully functional dependent on the primary key.

Example: Consider Product table

Product

| productID | product | brand |
| --- | --- | --- |
| 1 | Monitor | Apple |
| 2 | Monitor | Samsung |
| 3 | Scanner | HP |
| 4 | Headphone | JBL |

Product table after 2NF will be,

Products

| productID | product |
| --- | --- |
| 1 | Monitor |
| 2 | Scanner |
| 3 | Headphone |

Brand

| brandID | brand |
| --- | --- |
| 1 | Apple |
| 2 | Samsung |
| 3 | HP |
| 4 | JBL |

ProductsBrand

| pbID | productID | brandID |
| --- | --- | --- |
| 1 | 1 | 1 |
| 2 | 1 | 2 |
| 3 | 2 | 3 |
| 4 | 3 | 4 |

3) Third Normal Form (3NF)

* The table should be in 2NF
* No transition dependency exists.

Example: Consider Course table.

Course

| course | department | lecturer |
| --- | --- | --- |
| Relational databases | CS | Jeremy |
| Cloud databases | CS | Jeremy |
| Algorithms | CS | James |
| Data Science | Mathematics | Jane |
| Calculus | Mathematics | John |

Functional dependencies in this table are:

course -> lecturer,department

lecturer -> department

Candidate key is {course}

The table is not in 3NF, because there is a transitive functional dependency from the

candidate key course to department.

The functional dependency lecturer -> department violates the rules for 3NF.

So the resulting tables after 3NF will be,

Lecturers

| lecturer | department |
| --- | --- |
| Jeremy | CS |
| James | CS |
| Jane | Mathematics |
| John | Mathematics |

Courses

| course | lecturer |
| --- | --- |
| Relational databases | Jeremy |
| Cloud databases | Jeremy |
| Algorithms | James |
| Data Science | Jane |
| Calculus | John |