GEEKSCOOP
DBMS

By

MAI GNANA GANAPATHY M

YOGESH KUMAR R

GOKUL R

Why concurrency ?

- The Temporary Update (or Dirty Read) Problem.
- The Lost Update Problem.
- The Incorrect Summary Problem

What is ACID ?

What is BASE ?

ACID – Atomicity Consistency Isolation Durability

BASE - Basically Available, Soft State, and Eventual Consistency

Atomicity

Atomicity involves the following two operations:

Abort: If a transaction aborts then all the changes made are not visible.

Commit: If a transaction commits then all the changes made are visible.

Consistency

- The consistent property of database states that every transaction sees a consistent database instance.
- The transaction is used to transform the database from one consistent state to another consistent state.

For example: The total amount must be maintained before or after the transaction.
1.Total before T occurs = 600+300=900
2.Total after T occurs= 500+400=900

Isolation

- It shows that the data which is used at the time of execution of a transaction cannot be used by the second transaction until the first one is completed.
- In isolation, if the transaction T1 is being executed and using the data item X, then that data item can't be accessed by any other transaction T2 until the transaction T1 ends.
- The concurrency control subsystem of the DBMS enforced the isolation property

Durability

- The durability property is used to indicate the performance of the database's consistent state. It states that the transaction made the permanent changes.
- They cannot be lost by the erroneous operation of a faulty transaction or by the system failure. When a transaction is completed, then the database reaches a state known as the consistent state. That consistent state cannot be lost, even in the event of a system's failure.
- The recovery subsystem of the DBMS has the responsibility of Durability property.

BASE

- **BASE** is an acronym for **Basically Available, Soft State, and Eventual Consistency**. It is a set of properties that are often used to describe NoSQL databases, in contrast to the ACID properties that are used to describe traditional relational databases.
- **Basically Available** means that the database is always available to respond to user requests, even if it cannot guarantee immediate access to all data. This may be achieved by replicating the data across multiple servers, or by using a distributed architecture.
- **Soft State** means that the data values may change over time, even if the database is not being updated. This is because the database may not have been able to propagate all of the changes to all of the replicas yet.
- **Eventually Consistent** means that, over time, all of the replicas will eventually converge to a consistent state. This is achieved through a process called eventual consistency.

Joins

- A Join operation combines related tuples from different relations, if and only if a given join condition is satisfied.

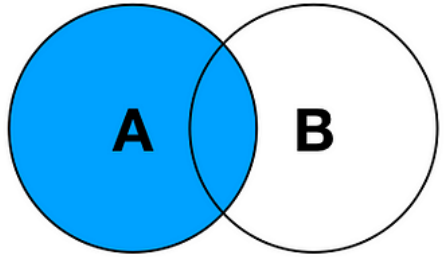
```sql
-- Join
SELECT *
FROM customers
JOIN orders ON customers.customer_id = orders.customer_id;

-- Subquery
SELECT *
FROM customers
WHERE customer_id IN (SELECT customer_id FROM orders);
```

Join vs Subquery

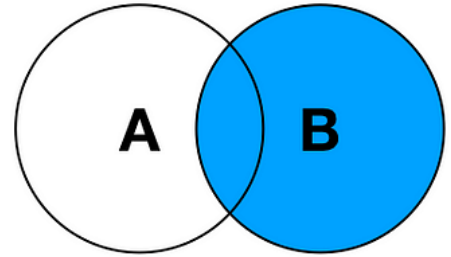
- **Performance:** Joins are generally faster than subqueries, because they can use indexes and other optimization techniques. Subqueries, on the other hand, may require more processing and memory, especially if they return large or complex results.
- **Readability:** Joins can be more readable than subqueries, especially for complex queries. Subqueries can be nested and difficult to follow, while joins can be expressed in a more straightforward way.
- **Maintainability:** Joins are easier to maintain than subqueries, because they are more explicit. Subqueries can be hidden in the WHERE or HAVING clause, which can make them difficult to find and debug.

There are some cases where subqueries may be necessary, such as when you need to return a single value or multiple values as part of the outer query

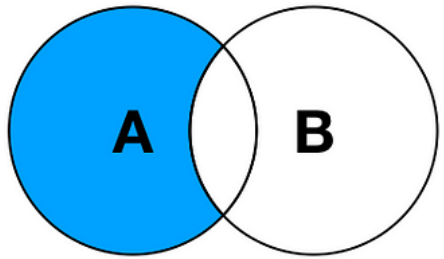
SQL JOINS



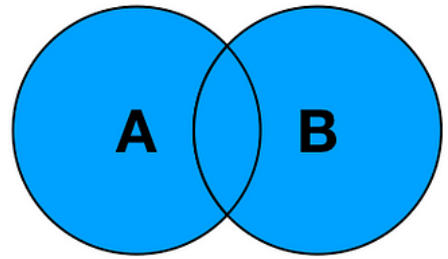
LEFT JOIN



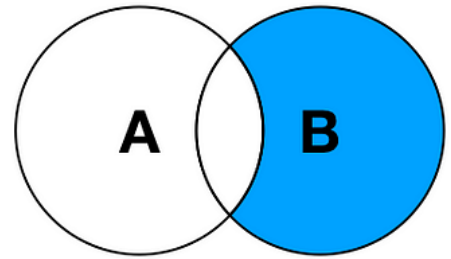
RIGHT JOIN



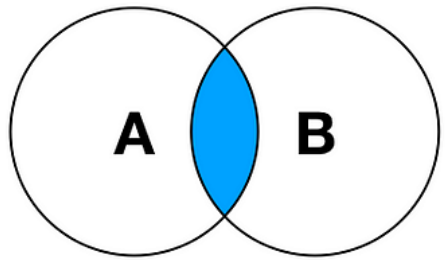
LEFT JOIN EXCLUDING
INNER JOIN



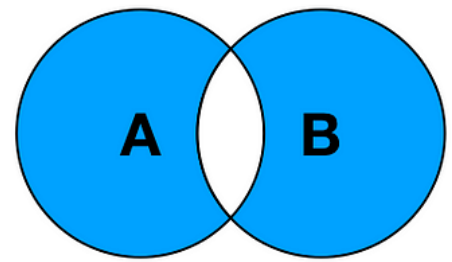
FULL OUTER JOIN



RIGHT JOIN EXCLUDING
INNER JOIN

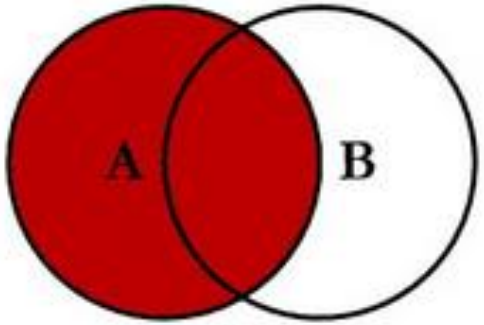


INNER JOIN

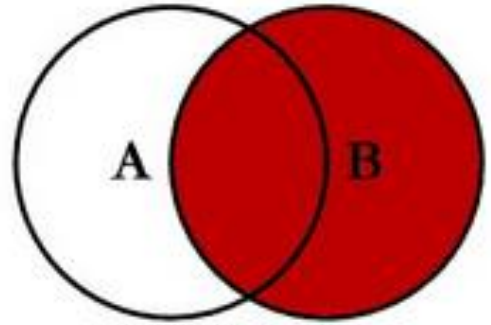


FULL OUTER JOIN EXCLUDING
INNER JOIN

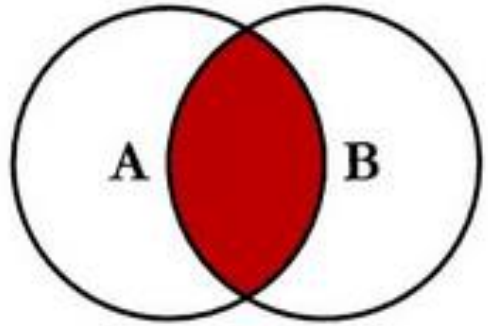
SQL JOINS



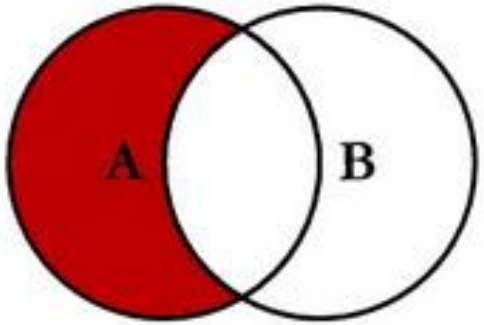

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



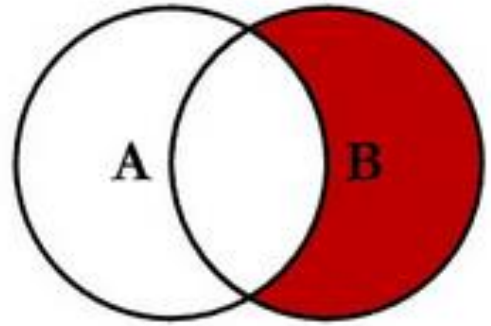

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



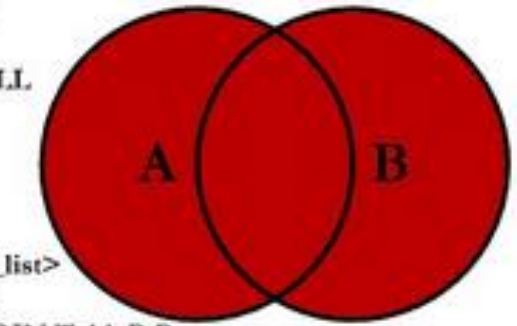

```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



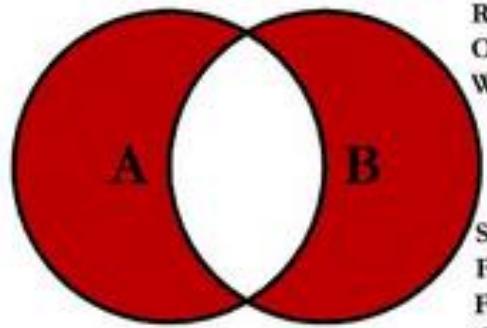

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```




```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

The Most Confusing Joins

Type of join	Definition	Example
Natural join	Joins two tables based on all columns with the same name and data type.	`SELECT * FROM customers NATURAL JOIN orders;`
Equi join	Joins two tables based on the equality of one or more columns.	`SELECT * FROM customers JOIN orders ON customers.customer_id = orders.customer_id;`
Cross join	Returns all possible combinations of rows from two tables.	`SELECT * FROM customers CROSS JOIN orders;`

Indexing

Indexing is a technique used to improve the performance of database queries. An index is a data structure that allows the database to quickly locate specific rows in a table.

Advantages

- **Faster query performance:** Indexes can significantly improve the performance of queries that filter, sort, or group data. This is because the database can use the index to quickly find the relevant rows, without having to scan the entire table.
- **Reduced resource usage:** Indexes can also reduce the amount of resources required to execute queries, such as CPU time and memory usage.
- **Improved scalability:** Indexes can help databases to scale more effectively, by allowing them to handle larger amounts of data and traffic.

Indexing

Example

```sql
-- Without index
SELECT * FROM customers WHERE country = 'USA';

-- With index
CREATE INDEX idx_customers_country ON customers (country);
SELECT * FROM customers WHERE country = 'USA';
```

DATABASE DESIGN