SUB-QUERIES, TRANSACTION, JOIN

DATABASE MANAGEMENT SYSTEM

SUJAN TAMRAKAR

Sub-queries

- A subquery is a SQL query nested inside a larger query.
- can be nested inside a SELECT, INSERT, UPDATE, or DELETE statement
- A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be followed.
- The inner query executes first before its parent query so that the results of an inner query can be passed to the outer query.
- A subquery is also called an inner query, while the statement containing a subquery is also called an outer query.

Sub-queries

- Subqueries must be enclosed within parentheses.
- Subqueries that return more than one row can only be used with multiple value operators such as the IN operator.
- Common use of subqueries is to perform tests for set **membership**, make set comparisons, determine set cardinality.
- The IN connective tests for set membership, where set is a collection of values produced by a select clause.
- The NOT IN connective is used to test for the absence of set membership.

SUB-QUERIES

- Finding all customers who have both a loan and an account at bank. [intersect operation]
  select distinct customerName from borrower where customerName in (select customerName from depositor);
- Above query proves that same query can be written in different forms allowing user to choose any preferred way
- Finding all customers who have a loan at the bank but do not have an account at the bank
  select distinct customerName from borrower where customerName not in (select customerName from depositor);

  select distinct customerName from borrower where customerName not in ('smith', 'john');

SUB-QUERIES

```
SELECT * FROM Customers WHERE age = (SELECT MIN(age)  FROM Customers );

SELECT customer_id, first_name FROM Customers WHERE customer_id IN (SELECT customer_id FROM Orders);

SELECT name, listed_price FROM paintings WHERE listed_price > (SELECT AVG(listed_price) FROM paintings );

SELECT first_name, last_name FROM collectors WHERE id IN ( SELECT collector_id FROM sales);
```

Transactions

- Consists of a sequence of query statements
- Transaction begins implicitly when an SQL statement is executed
- Sql statements must end the transaction either with ‘commit’ or ‘rollback’
- Commit
  - compels/saves the current transaction.
  - Makes the updates performed by the transaction become permanent in the db
  - After the transaction is committed, a new transaction is automatically started
- Rollback
  - Causes the current transaction to be rolled back.
  - It undoes all the updates performed in the transaction.
  - Db state is restored to what it was before the first statement of transaction executed.

Transactions

- Transaction rollback is useful if some error condition is detected during execution of a transaction.
- Once a transaction has executed commit work, its effects can no longer be undone by rollback method.
- Db system ensures that in cases of system crash or power outage or some failure, a transaction's effects will be rolled back if it has not yet executed 'Commit'.
- Ex: to transfer money from one account to another, we need to update two account balances. The 2 update statements would form a transaction. An error while a transaction executes one of its statements would result in undoing of the effects of earlier statements of the transaction, so that the db is not left in a partially updated state.

TRANSACTIONS - EXAMPLES

- Ex:

```
START TRANSACTION;
SELECT @A:=SUM(salary) FROM table1 WHERE type=1;
UPDATE table2 SET summary=@A WHERE type=1;
COMMIT;
```

```
START TRANSACTION;
INSERT INTO Test VALUES (5), (6);
INSERT INTO Test VALUES (7), (8);
ROLLBACK;
```

```
BEGIN TRANSACTION
    INSERT INTO EmployeeRecords (EmpID, FirstName, LastName, Education, Occupation, YearlyIncome, Sales) VALUES (5, 'SQL', 'Server', 'Education', 'Teaching', 10000, 200)
    UPDATE EmployeeRecords SET Education = 'Tutorials', YearlyIncome = 98000 WHERE EmpID = 5
    COMMIT TRANSACTION
BEGIN TRANSACTION
    INSERT INTO EmployeeRecords (EmpID, FirstName, LastName, Education, Occupation, YearlyIncome, Sales)  VALUES (7, 'SQL Server', 'Tutorial', 'Masters', 'Learn', 55000, 1250)
    SELECT * FROM EmployeeRecords
    ROLLBACK TRANSACTION
    SELECT * FROM [dbo].[EmployeeRecords]
```

JOIN

- Join operation denoted by ⋈
- Used to combine related tuples from two or more relations into single relation
- Important as it allows to process relationships among relations

loaNo	branchName	Amount		CustomerName	LoanNo
L170	Downtown	3000		Jones	L170
L230	Redwood	4000		Smith	L230
L260	Perryridge	1700		Hayes	L155
Loan				Borrower	

- Loan inner join borrower on loan.loanNo = borrower.loanNo gives following result:

Join condition

loanNo	branchName	Amount	CustomerName	LoanNo
L170	Downtown	3000	Jones	L170
L230	Redwood	4000	Smith	L230

JOIN

- The attributes of the result consist of the attributes of the left hand side relation followed by the right hand side relation.
- Alias can be used for the duplicate columns naming
- Ex:

```
SELECT column_name(s)
FROM table1
INNER JOIN table2 ON table1.column_name = table2.column_name;
```

- Types:
  - Natural inner join
  - Outer join
    - Left outer join (⟕)
    - Right outer join (⟖)
    - Full outer join (⟗)

JOIN

Natural inner join:

loan natural inner join borrower

- Computes the natural join of two relations
- This operation will result in reduction of repeated column.
- From previous query, the attribute loan_no appears only once in the result of the natural join

loanNo	branchName	Amount	CustomerName
L170	Downtown	3000	Jones
L230	Redwood	4000	Smith

JOIN

Outer join:

- An extension of the join operation to deal with missing information
- We can use outer join operation to avoid the loss of information
- Its 3 forms compute the join and add extra tuples to the result of the join

Forms / Types:

Left outer join

Right outer join

Full outer join

JOIN

Left outer join

- Takes all tuples from the left relation that did not match with any tuple in the right relation,
- pads the tuples with null values for all other attributes from right relation
- and adds them to the result of natural join (no repeat of column).

loanNo	branchName	Amount	CustomerName
L170	Downtown	3000	Jones
L230	Redwood	4000	Smith
L260	Perryridge	1700	null
Result of loan left outer join borrower			

JOIN

Right outer join

- Symmetric with left outer join
- Pads tuples from right relation that did not match any from the left relation with nulls
- Adds them to the result of the natural join
- All info from right relation is present in the result of the right outer join.

loanNo	branchName	Amount	CustomerName
L170	Downtown	3000	Jones
L230	Redwood	4000	Smith
L155	null	null	Hayes

Result of loan right outer join borrower

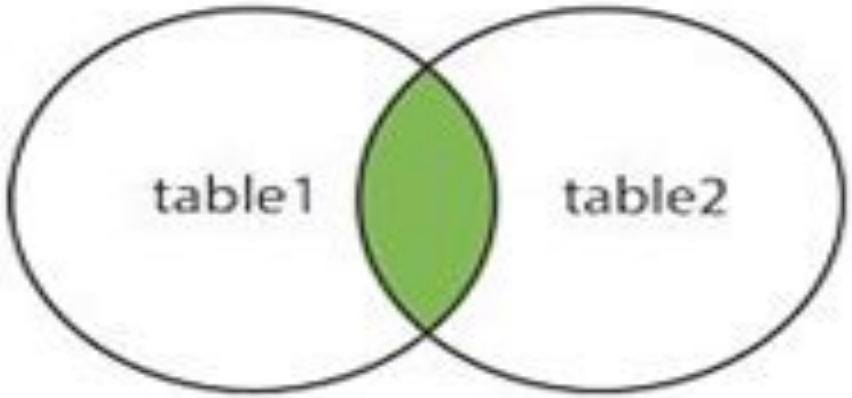
JOIN

Full outer join

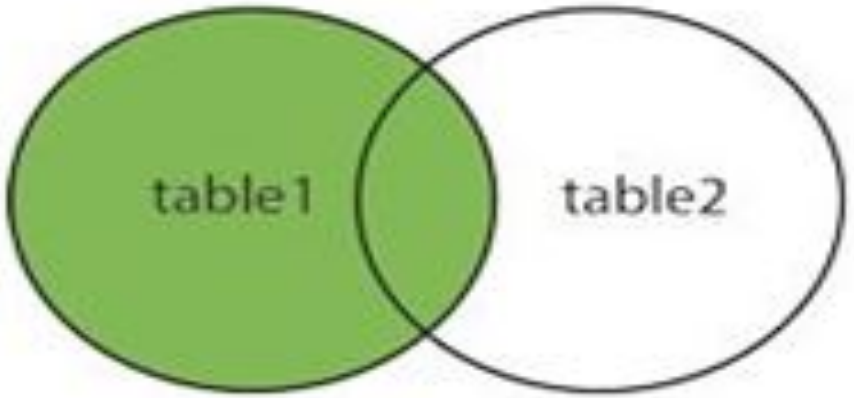
- Does both the operations.
- Pads tuples from the left relation that did not match any from the right relation as well as tuples from the right relation that did not match any from the left relation
- Adds them to the result of the join.

loanNo	branchName	Amount	CustomerName
L170	Downtown	3000	Jones
L230	Redwood	4000	Smith
L260	Perryridge	1700	null
L155	null	null	Hayes
Result of loan full outer join borrower			

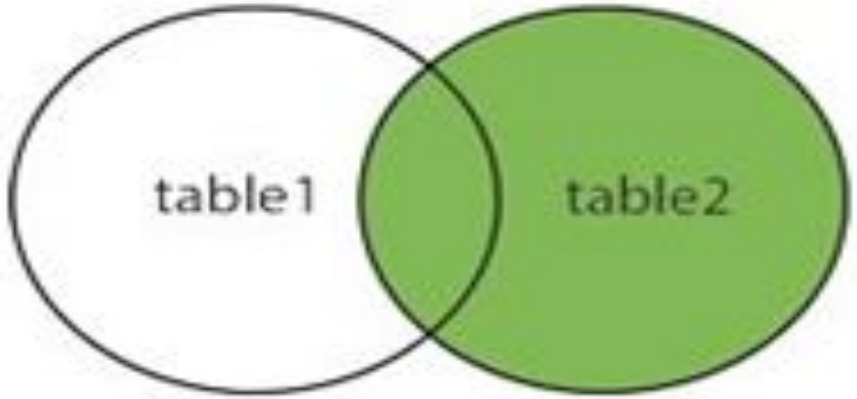
INNER JOIN



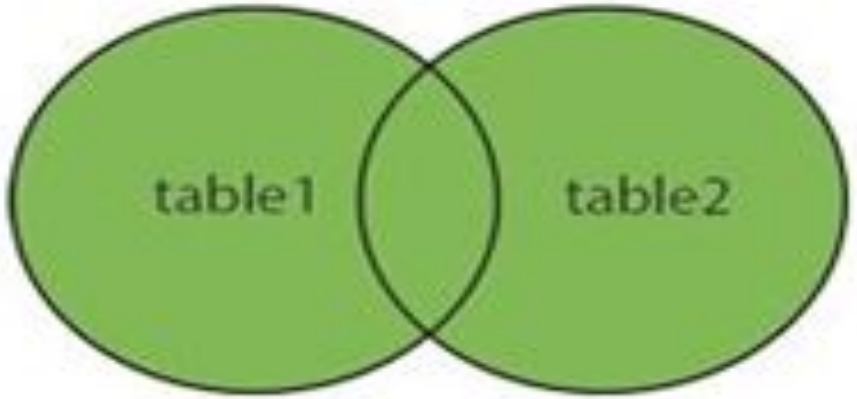
LEFT JOIN



RIGHT JOIN



FULL OUTER JOIN



Examples:

```
SELECT product_name, category_name, list_price FROM production.products p
INNER JOIN production.categories c ON c.category_id = p.category_id;

SELECT product_name, order_id FROM production.products p
LEFT JOIN sales.order_items o ON o.product_id = p.product_id;

SELECT product_name, order_id FROM sales.order_items o
RIGHT JOIN production.products p ON o.product_id = p.product_id;

SELECT M.name member, p.title project FROM pm.members m
FULL OUTER JOIN pm.projects p ON p.id = m.project_id;
```

Thank You!