1. [Sub-query](#Subquery)
2. [Normalization](#Normalization)
3. [Joining](#Joining)
4. [View](#View)
5. [Relational Algebra](#RelationalAlgebra)

**Sub-query**

**Using a Subquery to Solve a Problem**

Graphical user interface, text, application

Description automatically generated with medium confidence

**Subqueries**

You can place the subquery in a number of SQL clauses:

* + WHERE clause
  + HAVING clause
  + FROM clause
  + Subqueries can be used with SELECT, UPDATE, INSERT, DELETE statements along with expression operator. It could be equality operator or comparison operator such as =, >, =, (Less than) <= and Like operator.
* A subquery is a query within another query. The outer query is called as **main query** and inner query is called as**subquery**.
* The subquery generally executes first, and its output is used to complete the query condition for the main or outer query.
* Subquery must be enclosed in parentheses.
* Subqueries are on the right side of the comparison operator.
* [ORDER BY](https://www.geeksforgeeks.org/sql-order-by/) command **cannot** be used in a Subquery. [GROUPBY](https://www.geeksforgeeks.org/sql-group-by/)command can be used to perform same function as ORDER BY command.
* Use single-row operators with singlerow Subqueries. Use multiple-row operators with multiple-row Subqueries.

**syntax:**

SELECT column\_name

FROM table\_name

WHERE column\_name *expression operator*

( SELECT COLUMN\_NAME from TABLE\_NAME WHERE condition );

**Note:** Comparison operators fall into two classes: single-row operators (>, =, >=, <, <>, <=) and multiple- row operators (IN, ANY, ALL). The subquery is often referred to as a nested SELECT, sub-SELECT, or inner SELECT statement. The subquery generally executes first, and its output is used to complete the query condition for the main or outer query.

A picture containing text

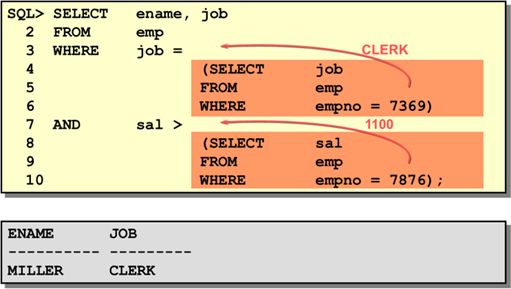
Description automatically generated

**Types of Subqueries**

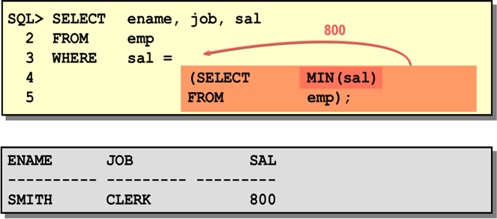
* + **Single-row subqueries:** Queries that return only one row from the inner SELECT statement
  + **Multiple-row subqueries:** Queries that return more than one row from the inner SELECT statement
  + **Multiple-column subqueries:** Queries that return more than one column from the inner SELECT statement.

**Example:**

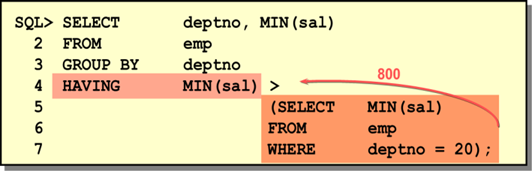
**Single Row Subquery:** Display the employees whose job title is the same as that of employee 7369.



**Group function:**



**Having class**

****

**Normalization**

***Normalization Definition:***

# In relational database design, the process of organizing data to minimize duplication.

# *Normalization* usually involves dividing a database into two or more tables and defining relationships between the tables.

# The objective is to isolate data so that additions, deletions, and modifications of a field can be made in just one table and then propagated through the rest of the database via the defined relationships.”

# "Normalization" refers to the process of creating an efficient, reliable, flexible, and appropriate "relational" structure for storing information. Normalized data must be in a "relational" data structure.

***Anomaly***

An error or inconsistency that may result when a user attempts to update a table that contains redundant data.

There are three types of Anomaly - **Insertion Anomaly, Deletion Anomaly, Modification Anomaly**

***Well Structure Relation***

A relation that contains minimal redundancy and allows users to insert, modify and delete the rows without error or inconsistencies.

Question – Is this a relation?

Answer – Yes: unique rows and no multivalued attributes

Question – What’s the primary key?

Answer – Composite: Emp\_ID, Course\_Title

***The Normal Forms:***

A series of logical steps to take to normalize data tables

* First Normal Form (1NF)
* Second Normal Form (2NF)
* Third Normal Form (3NF)

***First Normal Form Rule:***

A relation that contains no multivalued Attributes.

***Second Normal Form Rule:***

A relation in First Normal Form in which every attribute in fully functionally dependent in the primary key or Partial Functional dependency should be removed.

***Partial Functional Dependency***

A functional dependency in which one or more non-key attribute are functionally dependent in part (but not all) of the primary key.

***Functional Dependency***

A constrain between two attribute or two sets of attributes.

***Third Normal Form Rule:***

A relation in Second Normal Form has no Transitive Dependency present.

***Transitive Dependency:*** ***A Functional Dependency between two (or more) non-key attributes.***

**Joining**

**Types of Joins**

There are two main types of join conditions:

* **Equijoins**

An **equi join** is a type of join that combines tables based on matching values in specified columns.

Please remember that:

* The column names do not need to be the same.
* The resultant table contains repeated columns.
* It is possible to perform an equi join on more than two tables.

**Syntax**

* SELECT \*
* FROM TableName1, TableName2
* WHERE TableName1.ColumnName = TableName2.ColumnName;
* -- OR
* SELECT \*
* FROM TableName1
* JOIN TableName2
* ON TableName1.ColumnName = TableName2.ColumnName;
* **Non-equijoins:**

NON EQUI JOIN performs a JOIN using comparison operator other than equal(=) sign like >, <, >=, <= with conditions

**Syntax**

SELECT \*

FROM table\_name1, table\_name2

WHERE table\_name1.column *expression operator* (>, <, >=, <=) table\_name2.column;

* **Outer joins**

In the outer join, we consider any of the tables completely or both such that the remaining fields that were unmatched in both the tables were kept NULL.

**Syntax**

**SELECT \* FROM**

student

**(LEFT, Right, Full) JOIN**

location

**ON**

student.student\_id = location.student\_id;

* **Self joins**

Self-Join considers the same table as another table and outputs the resultant table after the required condition satisfies.

**Syntax**

**SELECT** s1.student\_id ,s1.student\_name **FROM**

student s1

**INNER JOIN**

student s2

**ON**

s1.student\_name= s2.student\_name AND s1.student\_id<> s2.student\_id;

**View**

**Simple Syntax:**

CREATE VIEW <view\_name> AS

SELECT <col>,<col> FROM <table\_name> WHERE <condition> ;

**Complex Syntax:**

CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW <view> [(alias[, alias]...)]

AS <subquery>

[WITH CHECK OPTION [CONSTRAINT constraint]] [WITH READ ONLY];

|  |  |
| --- | --- |
| **OR REPLACE** | re-creates the view if it already exists |
| **FORCE** | creates the view regardless of whether or not the base tables exist |
| **NOFORCE** | creates the view only if the base tables exist (This is the default.) |
| **View** | is the name of the view |
| **Alias** | specifies names for the expressions selected by the view’s query (The number of aliases must match the number of expressions selected by the  view.) |
| **subquery** | is a complete SELECT statement (You can use aliases for the columns in the  SELECT list.) |
| **WITH CHECK**  **OPTION** | specifies that only rows accessible to the view can be inserted or updated |
| **constraint** | is the name assigned to the CHECK OPTION constraint |
| **WITH READ ONLY** | ensures that no DML operations can be performed on this view |

**Removing a View**

**Syntax:** *DROP VIEW <view\_name>;*

**Inline Views**

* An inline view is a sub query with an alias (correlation name) that you can use within a SQL statement.
* An inline view is similar to using a named sub query in the FROM clause of the main query.
* An inline view is not a schema object.

**Relational Algebra**

Relational algebra is a procedural query language. It gives a step-by-step process to obtain the result of the query. It uses operators to perform queries.

* **Six basic operators**
  + select: σ
  + project: ∏
  + union: ∪
  + set difference: *–*
  + Cartesian product: x
  + rename: *ρ*

**Select Operation:**

* Notation: *σ* *p*(*r*)
* *p* is called the **selection predicate**
* Defined as:  
    
   *σp*(***r***) = {*t* | *t* ∈ *r* **and** *p(t)*}

Where *p* is a formula in propositional calculus consisting of **terms** connected by : ∧ (**and**), ∨ (**or**), ¬ (**not**)  
Each **term** is one of:

<attribute> *op* <attribute> or <constant>

where *op* is one of: =, ≠, >, ≥. <. ≤

* Example of selection:  
    
   σ *branch\_name=“Perryridge”*(*account*)

**Select Operation Example:**

*D*

Relation r

*A*

*B*

*C*

*α*

*α*

*β*

*β*

*α*

*β*

*β*

*β*

*1*

*5*

*12*

*23*

*7*

*7*

*3*

*10*

σA=B ^ D > 5 (r)

*A*

*B*

*C*

*α*

*β*

*α*

*β*

*1*

*23*

*7*

*10*

Diagram

Description automatically generated with medium confidence

**Project Operation:**

* **Notation:** 
  + **where *A1, A2* are attribute names and *r* is a relation name.**
* **The result is defined as the relation of *k* columns obtained by erasing the columns that are not listed**
* **Duplicate rows removed from result, since relations are sets**
* **Example: To eliminate the *branch\_name* attribute of *account*  
    
   ∏*account\_number, balance* (*account*)**

**Project Operation Example:**

Relation *r*:

*A*

*B*

*C*

*α*

*α*

*β*

*β*

*10*

*20*

*30*

*40*

*1*

*1*

*1*

*2*

*A*

*C*

*α*

*α*

*β*

*β*

*1*

*1*

*1*

*2*

=

*A*

*C*

*α*

*β*

*β*

*1*

*1*

*2*

**What is Sequence**

∏A,C (*r*)

### **Composition of Relational Operations**

### Find the customer who live in Harrison

### ∏*customer\_name* (σ *customer\_city=”Harrison”* (*customer*))

### Notice that instead of giving the name of a relation as the argument of the projection operation, we give an expression that evaluates to a relation

**Union Operation:**

* Notation: *r* ∪ *s*
* Defined as:
  + - *r* ∪ *s* = {*t* | *t* ∈ *r* or *t* ∈ *s*}
* For *r* ∪ *s* to be valid.
* *r,* *s* must have the *same* arity (same number of attributes)
  + 2. The attribute domains must be compatible (example: 2nd column   
     of *r* deals with the same type of values as does the 2nd   
     column of *s*)
* Example: to find all customers with either an account or a loan  
   ∏*customer\_name* (*depositor*) ∪ ∏*customer\_name* (*borrower)*

### Union Operation Example:

Relations *r, s:*

*A*

*B*

*α*

*α*

*β*

*1*

*2*

*1*

*A*

*B*

*α*

*β*

*2*

*3*

*r*

*s*

*A*

*B*

*α*

*α*

*β*

*β*

*1*

*2*

*1*

*3*

**σ**(r ∪s):