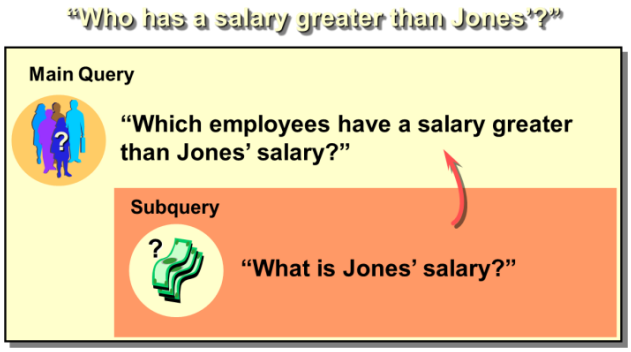
1. Sub-query
2. Normalization
3. Joining
4. View
5. Relational Algebra

Sub-query

Using a Subquery to Solve a Problem



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 command **cannot** be used in a Subquery. GROUPBY 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.

```
SQL> SELECT ename
  2  FROM   emp
  3  WHERE  sal >
  4                (SELECT sal
  5                 FROM   emp
  6                 WHERE  empno=7566);
```

2975

```
ENAME
----------
KING
FORD
SCOTT
```

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.

```
SQL> SELECT   ename, job
  2  FROM     emp
  3  WHERE    job =
  4                 (SELECT     job
  5                  FROM       emp
  6                  WHERE      empno = 7369)
  7  AND      sal >
  8                 (SELECT     sal
  9                  FROM       emp
 10                  WHERE      empno = 7876);
```

CLERK

1100

```
ENAME      JOB
---------- ---------
MILLER     CLERK
```

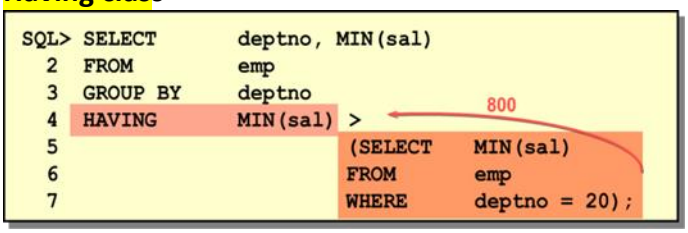
Group function:

```
SQL> SELECT   ename, job, sal
  2  FROM     emp
  3  WHERE    sal =
  4                 (SELECT     MIN(sal)
  5                  FROM       emp);
```

800

```
ENAME      JOB              SAL
---------- --------- ---------
SMITH      CLERK            800
```

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.

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: $\prod$
  - union: $\cup$
  - set difference: –
  - Cartesian product: x
  - rename: ρ

Select Operation:

- Notation: $\sigma_p(r)$
- p is called the **selection predicate**
- Defined as:

$\sigma_p(\boldsymbol{r}) = \{t \mid t \in r \textbf{ and } p(t)\}$

Where p is a formula in propositional calculus consisting of **terms** connected by : $\wedge$ (**and**), $\vee$ (**or**), $\neg$ (**not**)
Each **term** is one of:

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

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

- Example of selection:

$\sigma_{branch\_name="Perryridge"}(account)$

Select Operation Example:

Relation r

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

$\sigma_{A=B \wedge D > 5}(r)$

A	B	C	D
α	α	1	7
β	β	23	10

Project Operation:

- **Notation:** $\sigma_{A=B \wedge D > 5}(r)$
  - **where A_1, A_2 are attribute nar… n name.**
- **The result is defined as the relation o… ing the columns that are not listed**
- **Duplicate rows removed from result, …**
- **Example: To eliminate the *branch_no…***

$\Pi_{account\_number,\ balance}$ **(*account*)**

A	B	C
α	α	1
β	β	23

A	B	C	D
α	α	1	7
β	β	23	10

Project Operation Example:

Relation *r*:

A	B	C
α	10	1
α	20	1
β	30	1

What is Sequence

$$\prod_{A,C}(r)$$

Composition of Relational Operations

- Find the customer who live in Harrison
  - $\prod_{customer\_name}(\sigma_{customer\_city="Harrison"}(customer))$
  - 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: $r \cup s$
- Defined as:
  - $r \cup s = \{t \mid t \in r \text{ or } t \in s\}$
- For $r \cup s$ 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

  $\prod_{customer\_name}(depositor) \cup \prod_{customer\_name}(borrower)$

Union Operation Example:

Relations *r, s*:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

$\sigma(r \cup s)$:

A	B
α	1
α	2
β	1
β	3