Session 1 : 17/07/2023

DDL - CREATE, ALTER DROP, TRUNCATE, RENAME, Defines the schema
DML - UPDATE, INSERT, DELETE, REPLACE, MERGE, Data Manipulation Language
DQL - SELECT
DCL - GRANT, REVOKE
TCL - COMMIT, ROLLBACK, SAVEPOINT

Column, Attribute, field : record, row, tuple in a Table

CRUD are DML, DQL Commands. Insert, Select, Update/Replace, Delete

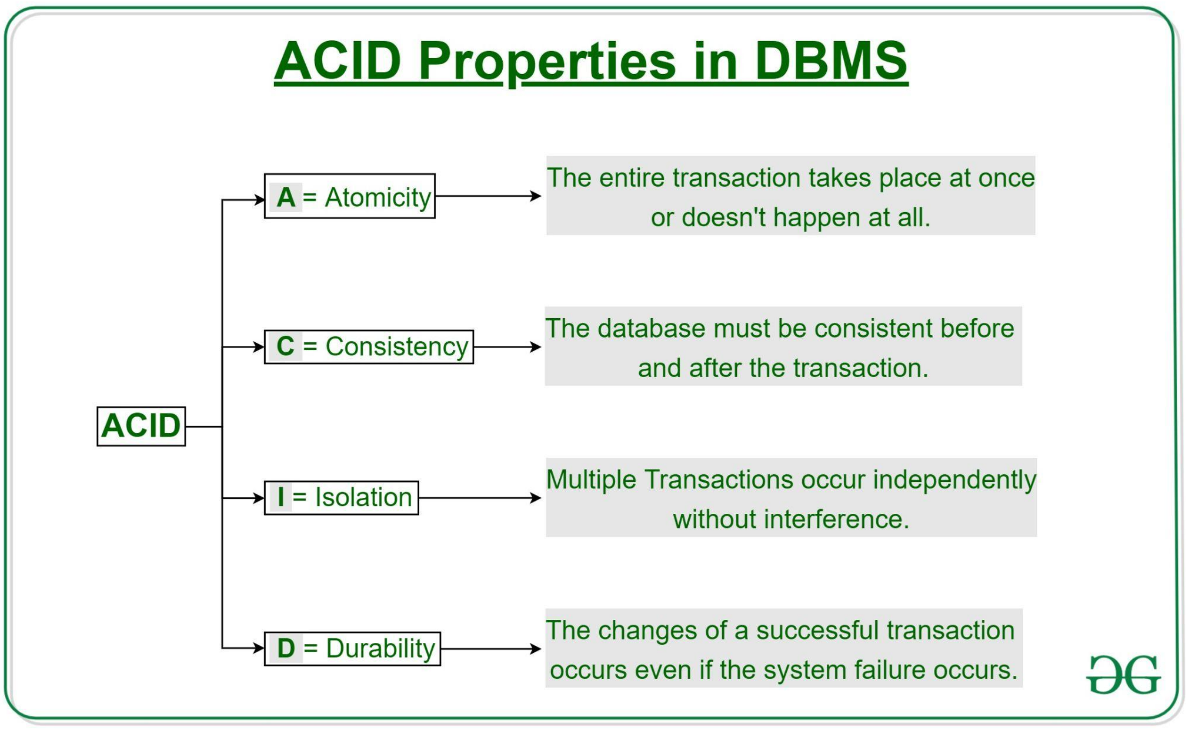
REPLACE is another command which will be in DDL

TRUNCATE vs DELETE
Truncate is DDL and Delete is DML. One deletes all the data, delete deletes only specified data.

Also, there is RENAME.

Logs : General Log tracks all user activities.

ACID properties:
Atomicity : An atomic Transaction(group of statements) has to be executed completely or none.
Consistency : Before and after transaction the database has to be consistent or valid state.Integrity constraints must be maintained so that the database is consistent before and after the transaction. maintaining data points in correct state after a transaction. Eg : Primary Key has to be followed in inserting new record. Even Manual Consistency can be maintained by maintaining the foreign, check constraints.
Isolation : keep transactions isolated until finished. There should be no interference between transactions. Failure or success of one transaction has to not interfere with other transaction.
Durability : changes become permanent once txn completed even if the system failure occurs. After a successful transaction, the database must be persistent.


ACID Properties in DBMS
ACID
A = Atomicity
The entire transaction takes place at once or doesn't happen at all.
C = Consistency
The database must be consistent before and after the transaction.
I = Isolation
Multiple Transactions occur independently without interference.
D = Durability
The changes of a successful transaction occurs even if the system failure occurs.

Levels of Abstraction:

1. Physical Level (Lowest Level) : How actually data is stored physically in secondary storage devices
2. Logical Level (Intermediate Level) : What data is stored.
3. External Level (High Level) View Level : Describes only part of the entire database.

Data Model :

A collection of concepts that can be used to describe the structure of a database-provides the necessary means to achieve this abstraction.

1. Relational Model: Table structure
2. Entity Relational Model : entity is real world object
3. Object Oriented
4. Semi structured Model

Database Languages :

Constraints :
- Domain Constraints : like age greater than 18 for electoral roll for age column
- Referential Integrity : foreign key
- Authorization : assigning roles
- Assertions : check statement

ER Diagram :
Attribute is set of properties to describe entity
Types of Attributes :
1. Simple
2. Composite

3. Single Values
4. Multi valued
5. Derived Attribute
6. Null Attribute

Keys:

1. Super Key
2. Candidate Key
3. Primary Key
4. Unique Key
5.

Data : Raw Facts -> Processed : Information -> lots : Knowledge -> lots : Wisdom

Data is a collection of facts, such as numbers, words, measurements, observations, or description of things.

Database is an organised collection of structured information, or data, typically stored electronically in a computer system.

Data management refers to professional practice of constructing, maintaining a framework, for storing, mining, and archiving the data integral to business.

DBMS is software to manage databases. Eg: MySQL, Oracle

Hierarchical Database : Data is organised in a tree format with root, child and parent nodes.
Network Database
Object Oriented DB : Data is represented in the form of objects. It is a combination of database concepts and OOP principles.

DBMS:

Interface to perform various operations like database creation, storing, updating, creating table in database.
It provides protection and security to the database. It also maintains data consistency in case of multiple users.

Tasks of DBMS:

1. Data Definition : creating, deleting databases
2. Data Updation : Insertion, modification and deletion of data.
3. Data Retrieval : Retrieving the data.
4. User Administration : maintain data integrity, enforcing data security, dealing with concurrent control, monitoring performance and recovering information

RDBMS :
Relational database is a type of database that stores and provides access to data points that are related to one another. These are based on relational model, an intuitive, straightforward way of representing data in models.

Difference between DBMS and RDBMS :

DBMS	RDBMS
DBMS applications store data as file	Store data in tabular form
Data is generally stored in hierarchical or navigational form	Tables have an identifier called primary key and data values are stored in form of tables
Normalization in not present in DBMS	It is.
Does not apply any security with regards to data manipulation.	Defines integrity constraint for the purpose of ACID properties.
Uses file system to store data, so there will be no relation between tables	Data values are stored in tabular format, so relationship between data values will be stored in the form of a table as well.
It has to provide some uniform methods to access stored information.	It supports a tabular structure of the data and a relationship between them to access the stored information.
It does not support distributed database	It supports.
It is meant to be for small organization and deal with small data. It supports single user.	It is designed to handle large amounts of data. It supports multiple users.
File systems, XML, etc..	MySQL, PostGRESQL, Oracle

Normalization :

Without Normalization ,
- Making relations large
- Not easy to maintain and update data
- Wastage and poor utilization of disk space and resources
- Likelihood of errors and inconsistencies increases.

Normalization is the process of decomposing the relations into relations with fewer attributes.

Anomalies that occur in relation:
Insertion anomaly
Deletion anomaly
Updation anomaly
Advantages of Normalization :
- It helps to minimize data redundancy
- Greater overall database organization
- Data Consistency within the database
- Much more flexible database design

- Enforces the concept of relational integrity

Normal Forms:
1NF : Atomic Values : single values
- Atomicity (No multiple Values)
- Datatypes of column should be of one type
- Order does not matter
- Two columns cannot have the same name.

Either a new column or new row can be added to make it 1NF
2NF : 1NF and No Partial Dependency : Depending on a part of candidate key
- Should be in 1NF
- Should not have any Partial Dependency. Partial Dependency refers to a situation in database normalization where a non-prime attribute (an attribute that is not part of the primary key) depends on only a part of the primary key, rather than on the entire key.

3NF : 2NF and no transitive functional dependency : Depending on a column which is not key
- Should be in 2NF
- Should not have Transitive Functional Dependency. A transitive functional dependency occurs when a non-prime attribute in a relation depends on another non-prime attribute, which itself depends on the primary key. In other words, the dependency "transits" through another attribute.

BC NF : 3NF and no functional dependency : super key determines each column

Disadvantages of Normalization :
1.

Session 2 : 18/07/2023

Subqueries can be used with select insert update and delete.

With Clause :

With average_salary(avg_Sal) as
(select cast(avg(salary) as int) from emp)
Select * from employee e, average_salary av
Where e.salary > av.avg_sal;

1. Find total sales for each store:
   Select s.store_id, sum(cost) as total_sales _per_Store
   From sales s
   Group by s.store_id
2. Find average sales of all stores
   Select cast(avg(total_sales_per_store) as int) as avg_Sales for_all_stores
   From (Select s.store_id, sum(cost) as total_sales _per_Store

From sales s
Group by s.store_id)

3. Find stores whose total sales > avg sales
   Select * from (Select s.store_id, sum(cost) as total_sales _per_Store
   From sales s
   Group by s.store_id) total_sales
   Join (Select cast(avg(total_sales_per_store) as int) as avg_Sales for_all_stores
   From (Select s.store_id, sum(cost) as total_sales _per_Store
   From sales s
   Group by s.store_id) x) avg_sales
   On total_sales.total_Sales_per_store > avg_sales.avg_sales_for_all_stores;

   Too complex queries, repeated queries can be minimised using with clause.
   The above can be changed as follows

```
With Total_Sales (store_id, total_sales_per_Store) as
        (Select s.store_id, sum(cost) as total_sales _per_Store
From sales s
Group by s.store_id),
        avg_Sales (avg_sales_for_all_stores) as
        (Select cast(avg(total_sales_per_store) as int) as avg_Sales for_all_stores
From Total_Sales)
Select
From Total_sales ts
Join avg_Sales av
On total_sales.total_Sales_per_store > avg_sales.avg_sales_for_all_stores;
```

Advantages of With Clause:
  Performance increases

When you have a subquery that has to be used every time (reusable), then it is ideal to use WITH clause

Above resource from TechTFQ yt channel

Temporary Tables:
  To store data temporarily.
  They are just similar to permanent database tables
  These are available from mysql version 3.23 but if you use an older version then can use heap tables.
  Temporary tables last as long as the session is alive.

Types of Temporary Tables:

1. Local Temporary Tables : accessible only in the session that has created it. A single hash # is used as a prefix of a table to create or drop. Random numbers are appended to the temporary tables

2. Global Temporary Tables : visible to all the connections and dropped when the last connection is closed. They must have a unique table name. There will be no random numbers suffixed at the end. To create, ## is used before the table name.

```
mysql> CREATE TEMPORARY TABLE SALESSUMMARY ( -> product_name
VARCHAR(50) NOT NULL -> , total_sales DECIMAL(12,2) NOT NULL DEFAULT
0.00 -> , avg_unit_price DECIMAL(7,2) NOT NULL DEFAULT 0.00 -> ,
total_units_sold INT UNSIGNED NOT NULL DEFAULT 0 );

mysql> INSERT INTO SALESSUMMARY -> (product_name, total_sales,
avg_unit_price, total_units_sold) -> VALUES -> ('cucumber', 100.25, 90,
2); mysql> SELECT * FROM SALESSUMMARY;

mysql> DROP TABLE SALESSUMMARY;
```

SQL Aggregate functions :

1. Count : SELECT COUNT(*) FROM PRODUCT_MAST;
2. Sum : Select sum(cost) from product;
3. Avg : select avg(cost) from product;
4. Max : select max(rate) from product;
5. Min : select min(rate) from product;

Pattern Matching :
It will be done using the LIKE clause.
% to represent zero one or more than one character
_ to represent a single character.

Session 3 : 19/07/2023

SQL Query Optimization :
Requirement For SQL Query Optimization:
- Enhancing Performance: The main reason for SQL Query Optimization is to reduce the response time and enhance the performance of the query. The time difference between request and response needs to be minimized for a better user experience.
- Reduced Execution Time: The SQL query optimization ensures reduced CPU time hence faster results are obtained. Further, it is ensured that websites respond quickly and there are no significant lags.
- Enhances the Efficiency: Query optimization reduces the time spend on hardware and thus servers run efficiently with lower power and memory consumption.

Best Practices For SQL Query Optimization
1. Use Where Clause instead of having
2. Avoid Queries inside a Loop
3. Use Select instead of Select *

4. Add Explain to the Beginning of Queries

    Explain <COMMAND>

5. Keep Wild cards at the End of Phrases
6. Use Exist() instead of Count()

Indexes :
An index is a schema object. It is used by the server to speed up the retrieval of rows by using a pointer.
Syntax:
```
CREATE INDEX index
 ON TABLE column;
```
For multiple columns:
Syntax:
```
CREATE INDEX index
 ON TABLE (column1, column2,.....);
```
Unique Indexes:
Syntax:
```
CREATE UNIQUE INDEX index
ON TABLE column;
```
When should indexes be created:

  A column contains a wide range of values.

  A column does not contain a large number of null values.

  One or more columns are frequently used together in a where clause or a join condition.

When should indexes be avoided:

  The table is small

  The columns are not often used as a condition in the query

  The column is updated frequently

DROP INDEX command.
Syntax:
```
DROP INDEX index;
```
Altering an Index: To modify an existing table’s index by rebuilding, or reorganizing the index.
```
ALTER INDEX IndexName
ON TableName REBUILD;
```
Confirming Indexes: You can check the different indexes present in a particular table given by the user or the server itself and their uniqueness.
Syntax:
```
select * from USER_INDEXES;
```
Renaming an index: You can use the system-stored procedure sp_rename to rename any index in the database.
Syntax:
```
EXEC sp_rename
  index_name,
  new_index_name,
  N'INDEX';
```

Pivot and Unpivot:

https://www.geeksforgeeks.org/pivot-and-unpivot-in-sql/
Pivot and Unpivot are relational operators that are used to transform one table into another for a more simpler view of table. Pivot operator converts the rows data of the table into the column data. The Unpivot operator does the opposite .Syntax:
1. Pivot:

```
SELECT (ColumnNames)
FROM (TableName)
PIVOT
 (
   AggregateFunction(ColumnToBeAggregated)
   FOR PivotColumn IN (PivotColumnValues)
 ) AS (Alias) //Alias is a temporary name for a table
```

2. Unpivot:

```
SELECT (ColumnNames)
FROM (TableName)
UNPIVOT
 (
   AggregateFunction(ColumnToBeAggregated)
   FOR PivotColumn IN (PivotColumnValues)
 ) AS (Alias)
```

Session 4 : 20/07/2023

ETL Process in Data Warehouse:
ETL stands for Extract, Transform, Load and it is a process used in data warehousing to extract data from various sources, transform it into a format suitable for loading into a data warehouse, and then load it into the warehouse.
ETL Tools: Most commonly used ETL tools are Hevo, Sybase, Oracle Warehouse builder, CloverETL, and MarkLogic.
Data Warehouses: Most commonly used Data Warehouses are Snowflake, Redshift, BigQuery, and Firebolt.

Cleaning Data in SQL:

```
coalesce(column, new value)
cast (column as dtype)
replace(dept_name, 'Information Technology', 'I.T')
date_part('month', CAST(birthdate AS date))
```

Islands and gaps :
https://www.red-gate.com/simple-talk/databases/sql-server/t-sql-programming-sql-server/introduction-to-gaps-and-islands-analysis/

Look for sections of JOINS, SUBQUERIES, STRING Functions and section 5 from udemy course.

Cast can also be as columnname :: text as shown in udemy

Session 5 : 21/07/2023

Recursive Queries : https://www.sqlshack.com/mysql-recursive-queries/
rank(), row_number(), dens_rank() window functions

Select rank() over(order by age) from customers;

Session 6 : 24/07/2023

A database has four main components:

1. Database schema
2. Database object definitions
3. Database data
4. User roles and privileges

Database DDL scripts include individual DDL scripts for database schemas and database objects. They contain:

Table and view definitions
Primary key definitions
Foreign key definitions
Stored procedures and functions definitions
Triggers definitions
Sequences definitions
Index definitions
Constraints definitions

DDL versioning

We can implement DDL versioning by applying the two main approaches:

1. Redefine all objects within the database, except for tables (tables are processed separately).
2. Update changed objects.

Session 7 : 25/07/2023

Common Table Expression:
A CTE (Common Table Expression) is a one-time result set that only exists for the duration of the query
The following is the basic syntax of CTE in SQL Server:

```
WITH cte_name (column_names)
AS (query)
SELECT * FROM cte_name;
```

It should keep in mind while writing the CTE query definition; we cannot use the following clauses:

1. ORDER BY unless you also use as TOP clause
2. INTO
3. OPTION clause with query hints
4. FOR BROWSE

The below syntax explains Multiple CTEs:

```
WITH
  cte_name1 (column_names) AS (query),
  cte_name2 (column_names) AS (query)
SELECT * FROM cte_name
UNION ALL
SELECT * FROM cte_name;
```

It is useful when we need to use ranking functions like ROW_NUMBER(), RANK(), and NTILE().

Types of CTE in SQL Server
SQL Server divides the CTE (Common Table Expressions) into two broad categories:

1. Recursive CTE
2. Non-Recursive CTE

Recursive CTE
A common table expression is known as recursive CTE that references itself. Its concept is based on recursion, which is defined as "the application of a recursive process or definition repeatedly."

In general, a recursive CTE has three parts:

1. An initial query that returns the base result set of the CTE. The initial query is called an anchor member.
2. A recursive query that references the common table expression, therefore, it is called the recursive member. The recursive member is union-ed with the anchor member using the `UNION ALL` operator.

3. A termination condition specified in the recursive member that terminates the execution of the recursive member.

The execution order of a recursive CTE is as follows:

- First, execute the anchor member to form the base result set (R0), use this result for the next iteration.
- Second, execute the recursive member with the input result set from the previous iteration (Ri-1) and return a sub-result set (Ri) until the termination condition is met.
- Third, combine all result sets R0, R1, … Rn using `UNION ALL` operator to produce the final result set.

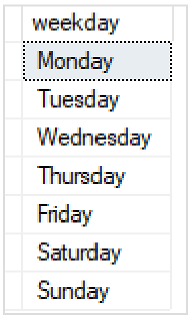
A) Simple SQL Server recursive CTE example

This example uses a recursive CTE to returns weekdays from `Monday` to `Saturday`:

```
WITH cte_numbers(n, weekday)
AS (
    SELECT
        0,
        DATENAME(DW, 0)
    UNION ALL
    SELECT
        n + 1,
        DATENAME(DW, n + 1)
    FROM
        cte_numbers
    WHERE n < 6
)
SELECT
    weekday
FROM
    cte_numbers;
```

Code language: SQL (Structured Query Language) (sql)

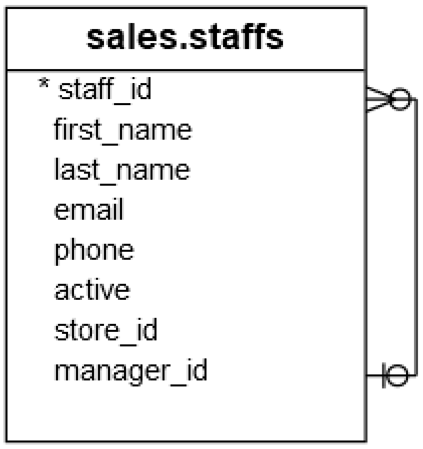
Here is the result set:



weekday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday

B) Using a SQL Server recursive CTE to query hierarchical data

See the following `sales.staffs` table from the sample database:



In this table, a staff reports to zero or one manager. A manager may have zero or more staffs. The top manager has no manager. The relationship is specified in the values of the `manager_id` column. If a staff does not report to any staff (in case of the top manager), the value in the `manager_id` is NULL.

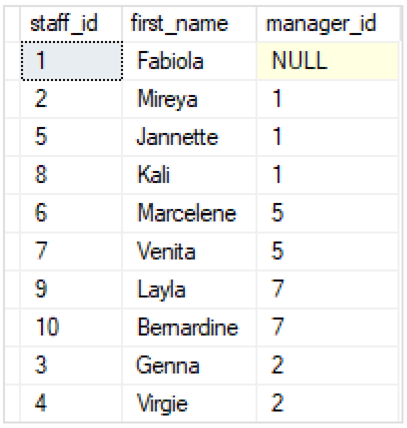
This example uses a recursive CTE to get all subordinates of the top manager who does not have a manager (or the value in the `manager_id` column is NULL):

```
WITH cte_org AS (
    SELECT
        staff_id,
        first_name,
        manager_id

    FROM
        sales.staffs
    WHERE manager_id IS NULL
    UNION ALL
    SELECT
        e.staff_id,
        e.first_name,
        e.manager_id
    FROM
        sales.staffs e
        INNER JOIN cte_org o
            ON o.staff_id = e.manager_id
)
SELECT * FROM cte_org;
```

Code language: SQL (Structured Query Language) (sql)

Here is the output:



staff_id	first_name	manager_id
1	Fabiola	NULL
2	Mireya	1
5	Jannette	1
8	Kali	1
6	Marcelene	5
7	Venita	5
9	Layla	7
10	Bernardine	7
3	Genna	2
4	Virgie	2

Non-Recursive CTE
A common table expression that doesn't reference itself is known as a non-recursive CTE.

Disadvantages of CTE
The following are the limitations of using CTE in SQL Server:

- CTE members are unable to use the keyword clauses like Distinct, Group By, Having, Top, Joins, etc.
- The CTE can only be referenced once by the Recursive member.
- We cannot use the table variables and CTEs as parameters in stored procedures.
- We already know that the CTE could be used in place of a view, but a CTE cannot be nested, while Views can.
- Since it's just a shortcut for a query or subquery, it can't be reused in another query.
- The number of columns in the CTE arguments and the number of columns in the query must be the same.

Window Functions:

Window functions applies aggregate and ranking functions over a particular window (set of rows). OVER clause is used with window functions to define that window. OVER clause does two things :

- Partitions rows into form set of rows. (PARTITION BY clause is used)

- Orders rows within those partitions into a particular order. (ORDER BY clause is used)

```
SELECT coulmn_name1,
 window_function(cloumn_name2)
 OVER([PARTITION BY column_name1] [ORDER BY column_name3]) AS
new_column
FROM table_name;


window_function= any aggregate or ranking function
column_name1= column to be selected
coulmn_name2= column on which window function is to be applied
column_name3= column on whose basis partition of rows is to be
done
new_column= Name of new column
table_name= Name of table
```

Ranking Window Functions :
Ranking functions are, RANK(), DENSE_RANK(), ROW_NUMBER()

RANK() –
As the name suggests, the rank function assigns rank to all the rows within every partition. Rank is assigned such that rank 1 given to the first row and rows having same value are assigned same rank. For the next rank after two same rank values, one rank value will be skipped.

DENSE_RANK() –
It assigns rank to each row within partition. Just like rank function first row is assigned rank 1 and rows having same value have same rank. The difference between RANK() and DENSE_RANK() is that in DENSE_RANK(), for the next rank after two same rank, consecutive integer is used, no rank is skipped.

ROW_NUMBER() –
It assigns consecutive integers to all the rows within partition. Within a partition, no two rows can have same row number.

Note –
ORDER BY() should be specified compulsorily while using rank window functions.

By Akash Das:
Import Export databases:
Commands and Steps

QEP : Query Execution Plan : used as EXPLAIN <query>

Query Pipeline :
(SELECT id, name) ⇒ 3 (FROM users) ⇒ 1 (WHERE id = 1;) ⇒ 2 ⇒ Order of execution

Primary Key, Unique Key are automatically indexed, where by there is no need for a full table scan for filtering data using where clause.

Session 8 : 26/07/2023

View:
a view is a virtual table based on the result-set of an SQL statement.
A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Note: A view always shows up-to-date data! The database engine recreates the view, every time a user queries it.

A view can be updated with the CREATE OR REPLACE VIEW statement.
SQL CREATE OR REPLACE VIEW Syntax

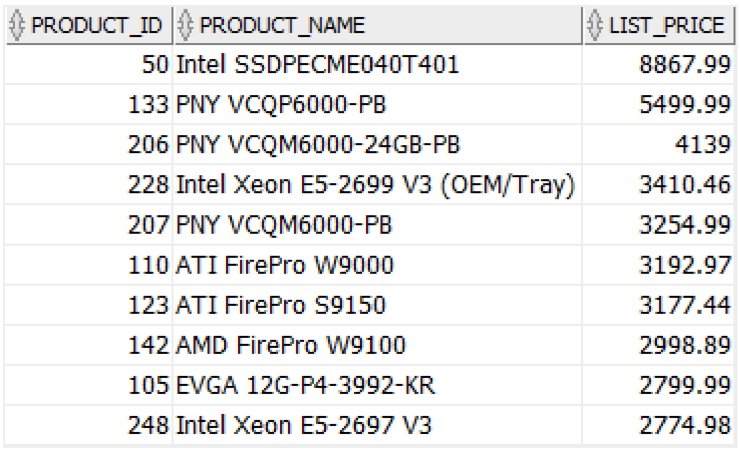
```
CREATE OR REPLACE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

```
DROP VIEW view_name;
```

Inline View :
An inline view is not a real view but a subquery in the FROM clause of a SELECT statement. The subquery specified in the FROM clause of a query is called an inline view. Because an inline view can replace a table in a query, it is also called a derived table.
A) simple Oracle inline view example
The following query retrieves the top 10 most expensive products from the products table:

```
SELECT
    *
FROM
    (
        SELECT
            product_id,
            product_name,
            list_price
        FROM
            products
        ORDER BY
            list_price DESC
    )
WHERE
    ROWNUM <= 10;
```

Code language: SQL (Structured Query Language) (sql)



PRODUCT_ID	PRODUCT_NAME	LIST_PRICE
50	Intel SSDPECME040T401	8867.99
133	PNY VCQP6000-PB	5499.99
206	PNY VCQM6000-24GB-PB	4139
228	Intel Xeon E5-2699 V3 (OEM/Tray)	3410.46
207	PNY VCQM6000-PB	3254.99
110	ATI FirePro W9000	3192.97
123	ATI FirePro S9150	3177.44
142	AMD FirePro W9100	2998.89
105	EVGA 12G-P4-3992-KR	2799.99
248	Intel Xeon E5-2697 V3	2774.98

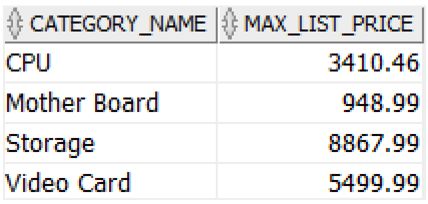
In this example, first, the inline view returns all products sorted by list prices in descending order. And then the outer query retrieves the first 10 rows from the inline view.

B) Inline view joins with a table example

The following example joins an inline view with a table in the FROM clause. It returns the product categories and the highest list price of products in each category:

```
SELECT
    category_name,
    max_list_price
FROM
    product_categories a,
    (
        SELECT
            category_id,
            MAX( list_price ) max_list_price
        FROM
            products
        GROUP BY
            category_id
    ) b
WHERE
    a.category_id = b.category_id
ORDER BY
    category_name;
```

Code language: SQL (Structured Query Language) (sql)



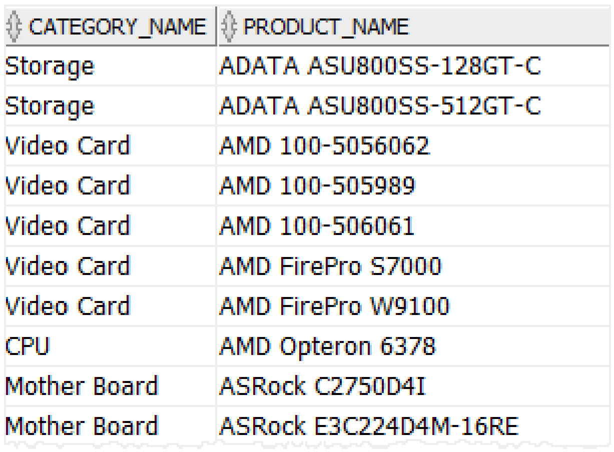
CATEGORY_NAME	MAX_LIST_PRICE
CPU	3410.46
Mother Board	948.99
Storage	8867.99
Video Card	5499.99

In this example, the inline view returns the category id list and the highest list price of product in each category. The outer query joins the inline view with the product_categories table to get the category name.
C) LATERAL inline view example
Consider the following statement:
```
SELECT
    category_name,
    product_name
FROM
    products p,
    (
        SELECT
            *
        FROM
            product_categories c
        WHERE
            c.category_id = p.category_id
    )
ORDER BY
    product_name;

Code language: SQL (Structured Query Language) (sql)
Oracle issued an error:
ORA-00904: "P"."CATEGORY_ID": invalid identifier
```

Code language: SQL (Structured Query Language) (sql)
This is because the inline view cannot reference the tables from the outside of its definition. Fortunately, since Oracle 12c, by using the LATERAL keyword, an inline view can reference the table on the left of the inline view definition in the FROM clause as shown in the following example:
```
SELECT
    product_name,
    category_name
FROM
    products p,
    LATERAL(
        SELECT
            *
        FROM
            product_categories c
        WHERE
            c.category_id = p.category_id
    )
ORDER BY
    product_name;
```

Code language: SQL (Structured Query Language) (sql)



CATEGORY_NAME	PRODUCT_NAME
Storage	ADATA ASU800SS-128GT-C
Storage	ADATA ASU800SS-512GT-C
Video Card	AMD 100-5056062
Video Card	AMD 100-505989
Video Card	AMD 100-506061
Video Card	AMD FirePro S7000
Video Card	AMD FirePro W9100
CPU	AMD Opteron 6378
Mother Board	ASRock C2750D4I
Mother Board	ASRock E3C224D4M-16RE

Note that the LATERAL inline views are subject to some restrictions listed in the documentation.
D) Oracle inline view: data manipulation examples
You can issue data manipulation statement such as INSERT, UPDATE, and DELETE against updatable inline view.
For example, the following statement increases the list prices of CPU products by 15%:

```
UPDATE
    (
        SELECT
            list_price
        FROM
            products
        INNER JOIN product_categories using (category_id)
        WHERE
            category_name = 'CPU'
    )
SET
    list_price = list_price * 1.15;
```

Code language: SQL (Structured Query Language) (sql)
And the following example deletes all video cards with the list price less than 1,000:

```
DELETE
    (
        SELECT
            list_price
        FROM
            products
        INNER JOIN product_categories
                USING(category_id)
        WHERE
            category_name = 'Video Card'
    )
WHERE
    list_price < 1000;

Code language: SQL (Structured Query Language) (sql)
```

.
By Akash Das:

```
select
      id
from t1
where id not in
      (select id from t2);

   -  To be completed from Notes
```

Session 9 : 27/07/2023

SQL Query Optimization Techniques:
Query optimization has many benefits, some of which are listed below.

- Minimize production issues: Every slow query requires high CPU, memory, and IOPS. We have seen most production database issues are caused by non-optimised queries.
- Performance issues: Slow query means slower response time of applications using the query, which results in poor end-user experience. It is very important to test logic/query with sufficient data before running it in production.
- Save infra cost: Unoptimised query requires more CPU, IOPS and memory. Additional load can be put on the same server if queries are optimised.

https://blogs.halodoc.io/learning-query-optimization-techniques/

By Akash Das

FOR, WHILE, REPEAT

Repeat is an exit controlled loop
WHILE is an entry controlled loop
FOR is implemented using
Looplabel , leave, iterate

Session 10 : 31/07/2023

String functions:
ASCII()
CHAR_LENGTH(), CHARACTER_LENGTH()
CONCAT()
CONCAT_WS()

…

Session 11 : 01/07/2023

Date Time Functions:
NOW() →
CURDATE()
CURTIME()
DATE() → extract date from datetime column
EXTRACT → Extract part from date time
Eg. EXTRACT(PART from date/time)
extract (DAY from ‘2021-01-01’) , month …

DATE_ADD() : DATE_ADD(DATE, INTERVAL value Unit_to_be_added);
DATE_SUB() :
DATE_DIFF()
DATE_FORMAT() → (‘2021-10-24 23:12:23’, “%W %D %M %Y %r”)

Session 12 : 03/07/2023

By Akash Das
View :

- Logical.
- Stored Query

Benefits :

1. Readability
2. Maintainability
3. Reusability

CREATE VIEW view_name AS query;

Types of views:

1. Update View
2. Read Only View
3. Materialized View
4. Inline View / Derived Table / Temporary Table

Cannot use View when integrating with Aggregate Functions, Set Operations, Sub Query

Example of Updatable view:

CREATE OR REPLACE VIEW customer_view as
SELECT CUSTOMERNUMBER, CUSTOMERnAME, PHONE, CITY FROM CUSTOMERS;

Read only view is not having any special keywords but can implement the functionality by implementing the Union,Subqueries, etc..

Session 13 : 04/07/2023

SQL Query Optimization Techniques:

1. **Indexing**: Unique and Primary key are indexed by default. Or create an index for a specific column of a table.
   Show Indexes from Orders;
   Drop index index_name on Orders
   Composite Index is an index on multiple columns
   Create index idxname on orders(c1, c2);
   But internally it creates two indexes
   Creating Indexes slows down DML queries but improves DQL query. So use it when having a lot of Read Operations.
2. **Use EXPLAIN:** The EXPLAIN keyword in MySQL is used to provide a detailed insight into how MySQL executes a query. It helps to understand the query execution plan and identify the bottlenecks.
3. ***Avoid SELECT** : Instead of using SELECT *, specify the columns that you want to retrieve to decrease the amount of data that needs to be read from the disk.
4. **Use LIMIT**: If you only need a certain number of rows, use the LIMIT keyword to avoid reading extra rows.
5. **Avoid Correlated Subqueries**: A correlated subquery is a subquery that depends on the outer query. It can result in the subquery being executed once for every row processed by the outer query. This can slow down your query significantly. It's often possible to rewrite correlated subqueries using joins.
6. **Use JOINs Instead of Subqueries:** If you need to combine data from multiple tables, use JOINs instead of subqueries as they are generally faster.
7. **Normalization and Denormalization:** Normalization reduces data redundancy but can lead to complex queries. Denormalization reduces the complexity of queries but may lead to data redundancy. Use a balance of both based on the specific use case.
8. **Data Types**: Use the most efficient data type possible. For example, INT is more efficient than DECIMAL or CHAR.
9. **Partitioning:** MySQL supports table partitioning, which can be a great way to optimize large tables.
10. **Caching**: MySQL has a built-in query cache. If enabled, it stores the result set of a SELECT statement, so if an identical statement is executed, the server can then retrieve the results from the cache rather than executing it again.
11. Union ALL is more performant than Union

Session 14 : 07/08/2023

Session 15: 08/08/2023

Rollup : aggregate of aggregates.
(c1, c2) → no of aggregates is n+1. (2+1 = 3) → (c1, c2), (c1), ()

Grouping:

It returns 1 when NULL in the orderyear column occurs in the super aggregate column.

Data Versioning : #36
If any data gets deleted or corrupted, data versioning allows us to revert back to the correct version. It helps in maintaining an audit trail for data. It provides the ability to analyse historical data.

—---------
MERGE INTO ?
—---------

Session 16 : 18/08/2023

Cursor :
Cursor is a Temporary Memory or Temporary Work Station. It is Allocated by Database Server at the Time of Performing DML(Data Manipulation Language) operations on the Table by the User. Cursors are used to store Database Tables. There are 2 types of Cursors: Implicit Cursors, and Explicit Cursors.

Four types of Cursor :
1. Read Only (this are non scrollable and non updatable)
2. Non Scrollable (this might be updatable or not)
3. Insensitive
4. Asensitive

Default cursor is readonly non scrollable
You can use MySQL cursor inside a stored routines.

Syntax :
- Declare a stored procedure
- Inside a stored procedure
    1. Declare variables for cursor
    2. Declare cursor : declare cursor for select statement
    3. Declare a NOT FOUND error handler : DECLARE CONTINUE HANDLER FOR NOT FOUND SET finished = 1; (Optional)
    4. Open <Cursor name>
    5. Fetch <Cursor name> into <variable list>
    6. Close <cursor name> (If not closed, slows down the performance)
    7. END Stored Procedure;

Cycle : DECLARE → OPEN → FETCH → until fetch is empty → CLOSE