Complex Database Queries with PostgreSQL

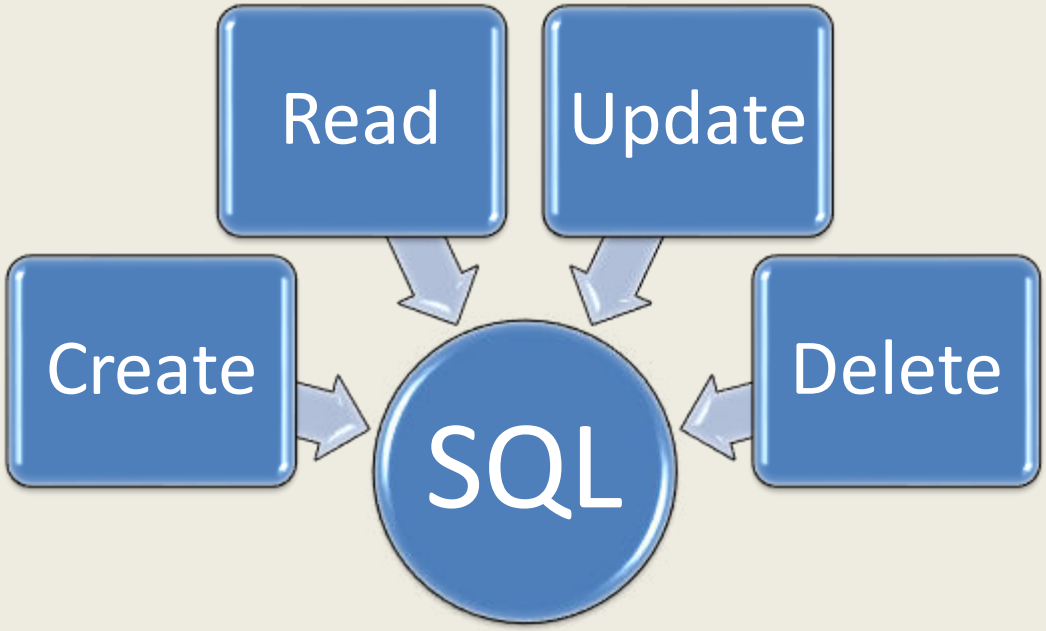
Dr.B.Hemalatha

Deputy Chief System Manager

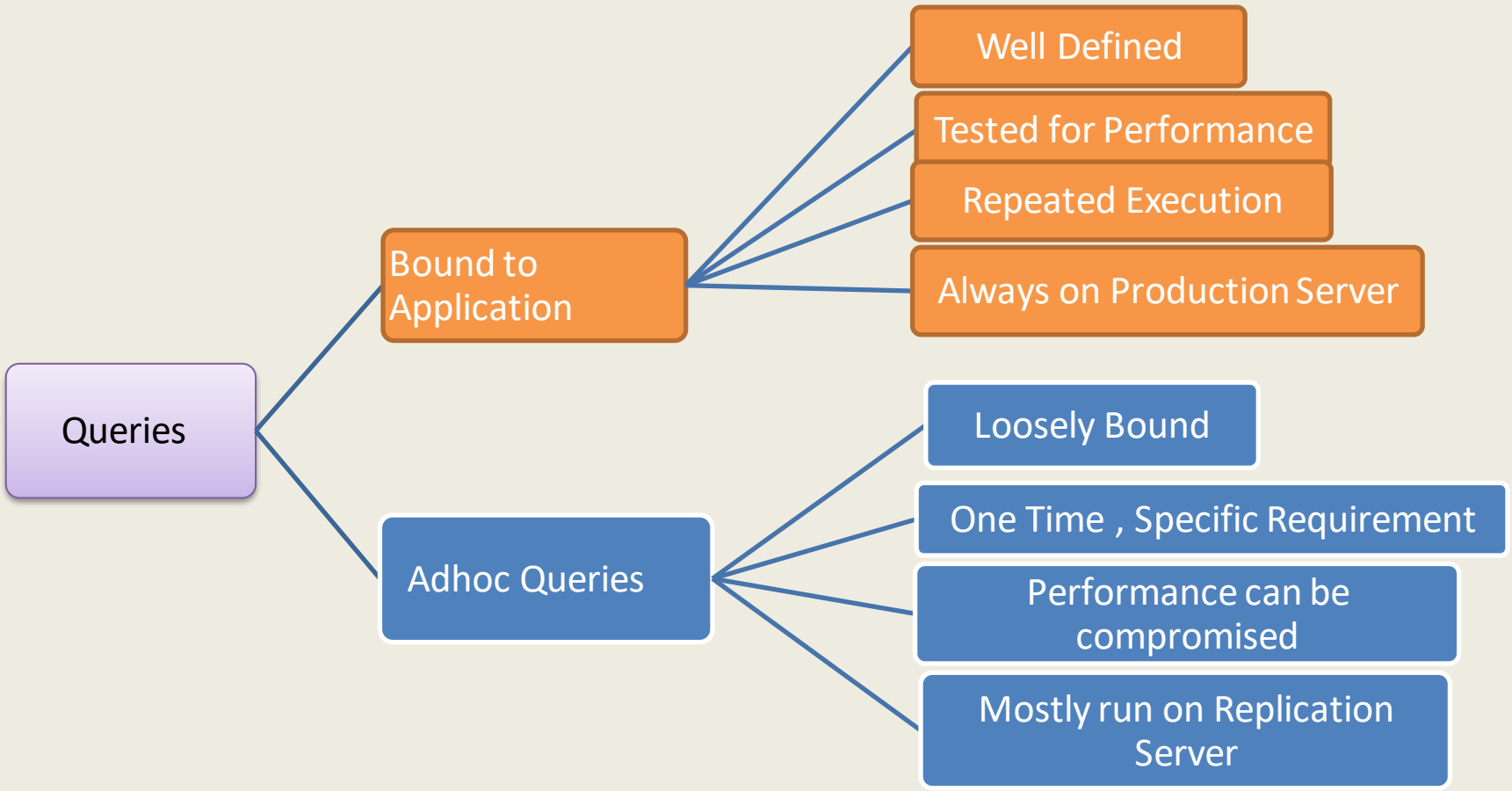
IIT Kharagpur-INDIA

SQL Queries

- Vital part of all applications
- Creating, retrieving and manipulating data in an efficient way.—CRUD



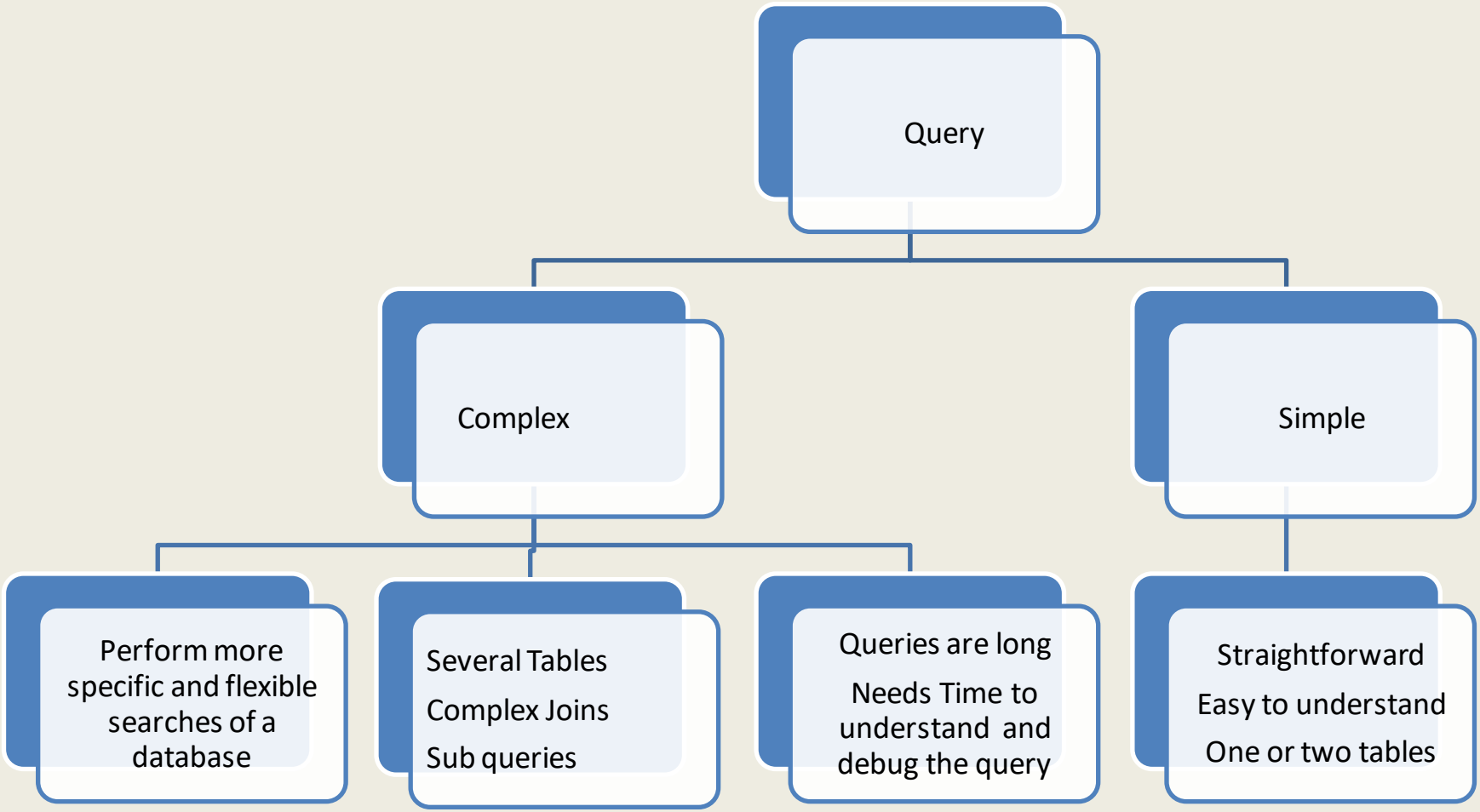
SQL Queries—Broad Classification



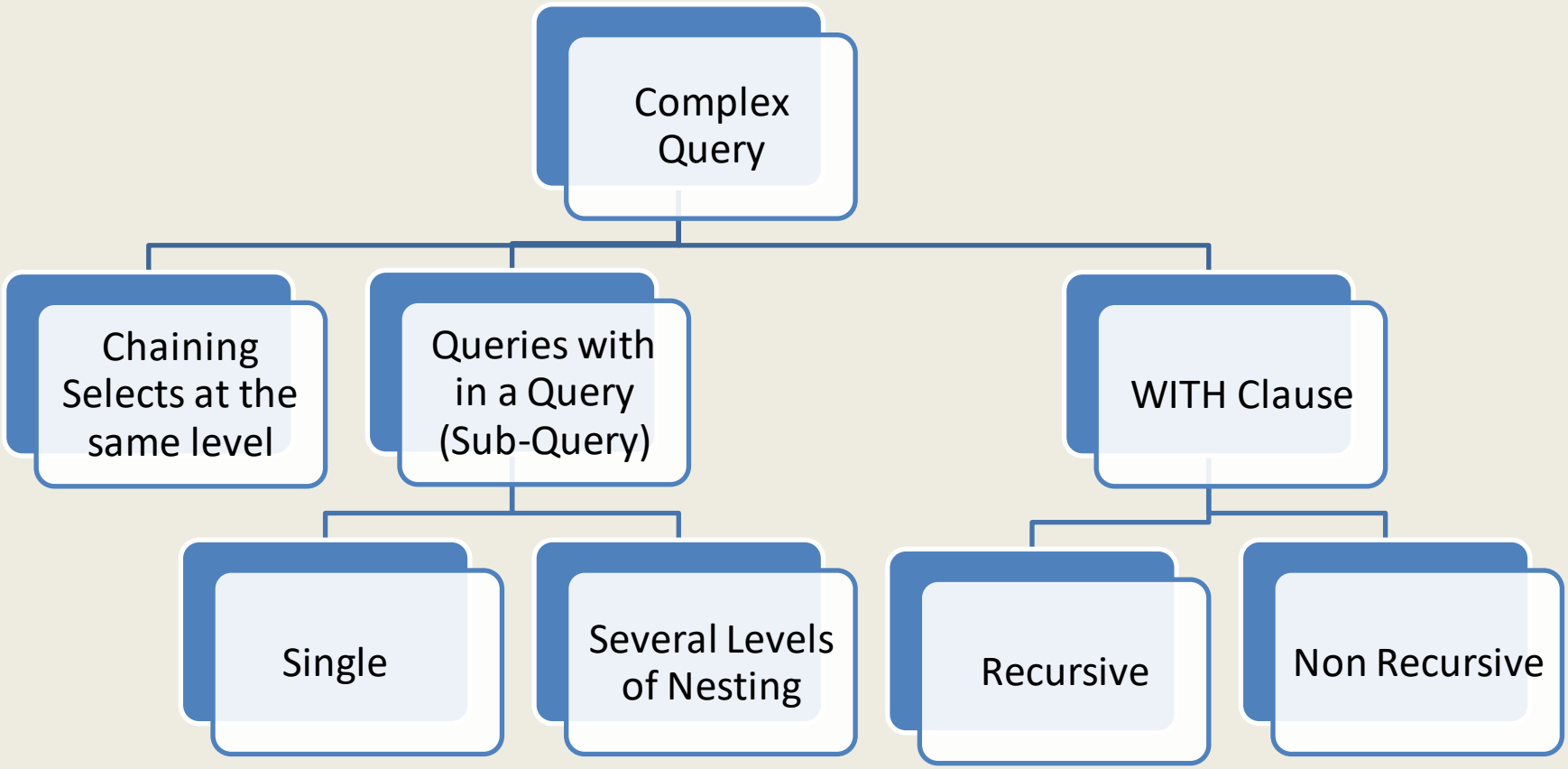
Adhoc Queries

- Adhoc queries are needed
  - To cater to the Data Requirements for Analytics
  - RTI requirements that are diverse and different
  - Specific requirements from National and International bodies
  - To test the output of a batch program
- More often the data requirement is in a particular format (pivot/un pivot)
  - In most cases these need complex queries to generate the data
- Adhoc queries work on data that will not normally change in that time span. Not currently transactional.
- Adhoc queries are run on a Streaming Replication Server using
  - pgAdmin Client
  - Microsoft Excel through ODBC drivers
  - Customized Application Interface
- Features available in PostGre 9.x and upwards make adhoc data retrieval easier

SQL Queries—Degree of Difficulty



Complex Queries--Construction



PostGreSQL and SQL Standard Conformance

- PostgreSQL tries to conform with the **SQL standard** where such conformance does not contradict traditional features or could lead to poor architectural decisions. Many of the features required by the SQL standard are supported, though sometimes with slightly differing syntax or function. Further moves towards conformance can be expected over time. As of the version 11 release in October 2018, PostgreSQL conforms to at least 160 of the 179 mandatory features for SQL:2011 Core conformance, where as of this writing, no relational database meets full conformance with this standard.

(Source: https://www.postgresql.org/about/)

A few PostGreSQL features for Complex Queries

- JSON Support
- Filter Clause –to subsets of data meeting certain conditions, used with aggregate and window functions
- Array Aggregates -Powerful way of converting several rows into one column
- Window Functions —Operates on a set of rows(OVER (PARTITION))
- Statistical Functions with the WITHIN GROUP clause
- Grouping Sets —Complex Grouping operations on all possible groups, rollup & cube (Multiple Group by in a single query)
- Common Table Expressions (CTE –WITH)
- Use of Temporary Functions (pg_temp)
- All the above features provide a elegant way of writing a complex query and delegating data fetching to the database

Platform & Examples

- All queries run & checked on pgAdmin4 version 3.2 on 64 bit Windows 10 Pro.
- Database "PostgreSQL 10.5 on x86_64-pc-linux-gnu, compiled by gcc (GCC) 4.8.5 20150623 (Red Hat 4.8.5-28), 64-bit“
- Representative Results only, real time object names and data have been masked.

JSON--Utility

- JSON Support -PostgreSQL provided JSON support since 9.2 and JSONB from 9.4 onwards. Postgre added the JSON feature well before the SQL standard added it in 2016 release.
- JSON objects particularly useful in complex queries to pivot and unpivot data.

Quick Examples-Unpivoting using JSON

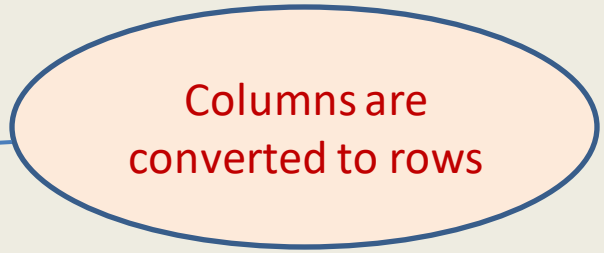
Choices
Regno
Choice1
Choice2
Choice3
--
Choicen
Other columns

Straight query: select regno, choice1, choice2,...choicen from choices

Unpivoting

```
select choices."regno" As regno,  j.key ,  j.val
from choices, LATERAL jsonb_each_text
(to_jsonb(choices)) AS j(key,val)
where  adm_year = 'current'  and j.key  ilike '%cho%'  and
j.key <> 'choice_allotted'  and coalesce(j.value,' ') not in ('')
order by regno, char_length(j.key), j.key
```

Regno	Key	Value
17XXX02	choice1	EE
17XXX02	choice2	ME
17XXX02	choice3	IE



Example-GROUPING SETS

```
select insti_scode, branch, count(distinct(regno))
from allocations where  allotted = 1 and session
= 'current ' group by  rollup(1,2)
```

Insti_scode	branch	count
IITKGP	AE	31
IITKGP	AG	28
IITKGP	AR	40
:		
IITKGP		1324

GRAND TOTAL

Branch wise count and grand total, 2 group by clauses

Filter Clause—Elegantly replaces CASE WHEN (pivots the output)

```
select branch, count(*) as tot, count(*) filter (where
gender = 'F') as female, count(*) filter (where gender= 'M')
as male,  from allotted where insti_scode = 'IITKGP'  and
allotted = 1  group by rollup(1)  order by 3 desc nulls first
```

branch	tot	female	male
	1587	263	1324
ME	181	27	154
GG	110	21	89
EE	118	18	100
EC	114	17	97

Overall Count

Branch wise count with gender segregation and overall count

Window Functions –Example using rank()

```
select * from (select a.roll,b.name,b.course,a.cgpa , rank() over
(partition by b.course order by a.cgpa desc) as rr from
performance a, studentmaster b where a.roll = b.roll and a.semno
= 'final' and a.session = 2016 and a.cgpa > 8.5 ) a where rr <=2
```

Roll	Name	Course	CGPA	Rank
13AEXXXX1	XXXXXXX	AE	9.02	1
13CHXXX33	XXXXXXX	CH	9.21	1
13CHXXX95	XXXXXXX	**CH**	**9.06**	2
13CHXXX01	XXXXXXX	**CH**	**9.06**	2
13CSXX110	XXXXXXX	CS	9.26	1

Window Functions –Cumulative Totals

```
select semno, sem_crd_taken, sem_crd_cleared,
sum(sem_crd_taken) over ( order by semno rows between
unbounded preceding and current row ), sum(sem_crd_cleared)
over ( order by semno rows between unbounded preceding and
current row )from performance where roll = 'xxxxxxx' group by
semno, sem_crd_taken,sem_crd_cleared
```

Semno	Taken	Cleared	Cum Taken	Cum Cleared
1	23	23	23	23
2	22	22	45	45
3	21	21	66	66
4	21	21	87	87
5	22	22	109	109
6	22	22	131	131
7	23	23	154	154
8	22	22	176	176

Cumulative Credits Taken and Cleared in a single query

Statistical Functions(Mean, Median, Mode in one go)

- select deptcode, round(avg(cgpa),2) , **mode() within group (order** by cgpa), **percentile_cont(0.5) within group (order by** cgpa), **percentile_disc(0.5) within group (order by** cgpa) from performance where semno = 'Final' and session = 'current' group by 1

Dept	Avg	Mode	MedianC	MedianD
AE	7.34	8.48	7.73	7.73
AG	6.92	8.18	7.36	7.36
AR	7.21	6.52	7.52	7.52
BT	7.2	6.22	7.3	7.3

Deptwise Mean, Median & Mode of CGPA

Statistical Functions –Comparison with Local and Global Average

```
select deptcode,roll, semno, cgpa,round(avg(cgpa) over (partition by deptcode order by deptcode) ,2) as depavg,
cgpa,round(avg(cgpa) over (),2) as overall_avg from performance
where year = 2018 and semno =1 group by 1,2,3,4 order by 1,2
```

Dept	Regno	Semno	CGPA	Depavg	Overall_avg
AE	18AEXXXX1	1	7.73	7.34	7.74
AE	18AEXXXX2	1	9.18	7.34	7.74
:	:	:	:	:	
AR	18ARXXXX4	1	8.09	7.21	7.74
AR	18ARXXXX5	1	6.55	7.21	7.74

Comparing individual scores with local and global averages

Array_agg functions—Subject Overlaps

```
select a.subno, count(distinct a.roll), array_agg(distinct b.subno order by b.subno) as
associations, count(distinct b.subno) as intersections from registration a, registration
b where a.subno in (select subno from subjects_master where subject_type = 'T') and
b.subno in (select subno from subjects_master where subject_type = 'T') and a.roll =
b.roll group by 1 order by 4 desc
```

Subno	Strength	Associations	Intersections
EP60020	300	{AE40008,AE40034,AE51007 …}	259
MA20104	753	{AE21002,AE21004,AE21008 …}	203
BM40002	104	{AE31002,AE31004,AE31006,	161
AI61002	142	{AE40008,AE40010,AE51007,	161
EP60002	137	{AE21002,AE21004,AE21008,AE40008,	156

Array of subjects having common registrations through array_agg function. Rows are converted to a single column

Temp Functions

```
create or replace function pg_temp.curr ( r VARCHAR)
//Temporary function to fetch the curricular details of a student
RETURNS TABLE ( r1  bpchar, cc  varchar,  sem  smallint,   patt1 text[], tot1 numeric)
AS $$
BEGIN
RETURN QUERY
(select roll, coursecode, maxsem, array_agg(subno order by subno) as patt, tot
from z   group by 1, 2,3 5 order by 1,2,3) ;
END; $$     LANGUAGE 'plpgsql';
```

- Gives a dynamic character
- Can be used like a relation
- Lasts as long as session exists

Temporary Functions in Query

- select * from **pg_temp.curr**(’13MEXXXX’)
- select *a*.*, b.name from studentmas b, **pg_temp.curr**(b.rollno) *a* where b.roll ilike '13ME%' --*Function used like a relation*

Roll	Coursecode	Max Semno	Pattern	Tot Sub
13MEXXX1	ME	8	{Br-1,De-50,El-6,HS-1,IE-1}	59
13MEXXX2	ME	8	{Br-1,De-50,El-6,HS-1,IE-1}	59

Common Table Expressions

- A common table expression (CTE) is as a temporary result set that is defined within the execution scope of a single SQL statement.
- A CTE is similar to a derived table and lasts only for the duration of the query.
- Unlike a derived table, a CTE can be self-referencing and can be referenced multiple times in the same query.

CTE--Explained

- Complex Query with nesting of subqueries

-*Select … from* (select … from (*select…from*) x) y

Outer Most Next Level Inner Most

-Reading the statement inside out

- CTE provides a compact way to break queries into manageable and comprehensible blocks

  – *WITH innermost as(select…), nextlevel as (select.. From t1, innermost), outermost as(select * from innermost, nextlevel where <>)*

  *select * from outermost ----Main Query*

CTE--Advantages

- Using a CTE offers the advantages of improved readability and ease in maintenance of complex queries. The query can be divided into separate, simple, logical building blocks. These simple blocks can then be used to build more complex, interim CTEs until the final result set is generated.
- CTEs are statement scoped views
- Non Recursive or Recursive
- Read Top down

CTE—Non recursive example

Grade Distribution and Subwise Grades through CTE

- WITH toppers as (select * from(select roll, dept,cgpa rank() over (partition by dept order by cgpa desc) as rr) x where rr<=2) , grade_distri as (select roll, dept,cgpa,count(*), count(*) filter (where grade = 'EX') … from toppers, performance group by roll,dept,cgpa), subwise_grades(select rollno, array_agg(subno||'-'grade order by grade) as patt from grades, toppers group by subno)
- select a.*, b.patt from grade_distri, subwise_grades

Roll	Dept	Cgpa	TOT	EX	A	B	C	D	P	F	Patt
13AE	AE	8.85	64	17	23	18	4	1	1	0	{AE21-A,...EV2-P}

CTE—Repeated Condition

- CTEs are useful when a condition has to be repeatedly applied in the where clause.
- WITH current_exam_session as (select subno, exam_session from *exam_timetable where session = 'current' and exam_type = 'mid'*) ,student_exam_timetable as (select roll, a.subno, exam_session from current_exam_session a, registration b where a.subno = b.subno), sub_wise_slot_occupancy as (select a.subno,a.exam_session,array_agg(distinct TRIM(b.exam_session) order by TRIM(b.exam_session)) as occupancy from student_exam_timetable a, student_exam_timetable b where a.roll = b.roll group by a.subno, a.exam_session)
- --Main Query

  select * from sub_wise_slot_occupancy

Subno	Exam Session	Occupancy
AI60001	S2	{S1,S2,S4,S5,S7}

CTE—Other Uses

- Assign column names to unnamed columns using CTE

```
WITH grademst(grade, points) as
( values ('A', 9), ('B',8),('C', 7),('D', 6), ('P',5),('F',0), (' ',0))
select grade, points from grademst
```

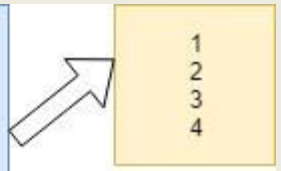
- Overload a schema table with CTE

Grade	Points
A	9
B	8
:	
	0

CTE-Recursive

- Refer to themselves in the second leg of UNION ALL
- Example:

```
WITH RECURSIVE t(n) AS (
VALUES (1)
UNION ALL
SELECT n+1 FROM t WHERE n <=3
)
SELECT * FROM t;
```



- Actually an iteration—similar top while <cond> loop
- Execution terminates when no tuples are returned.
- To prevent an infinite loop, limit clause can be used in the outer select

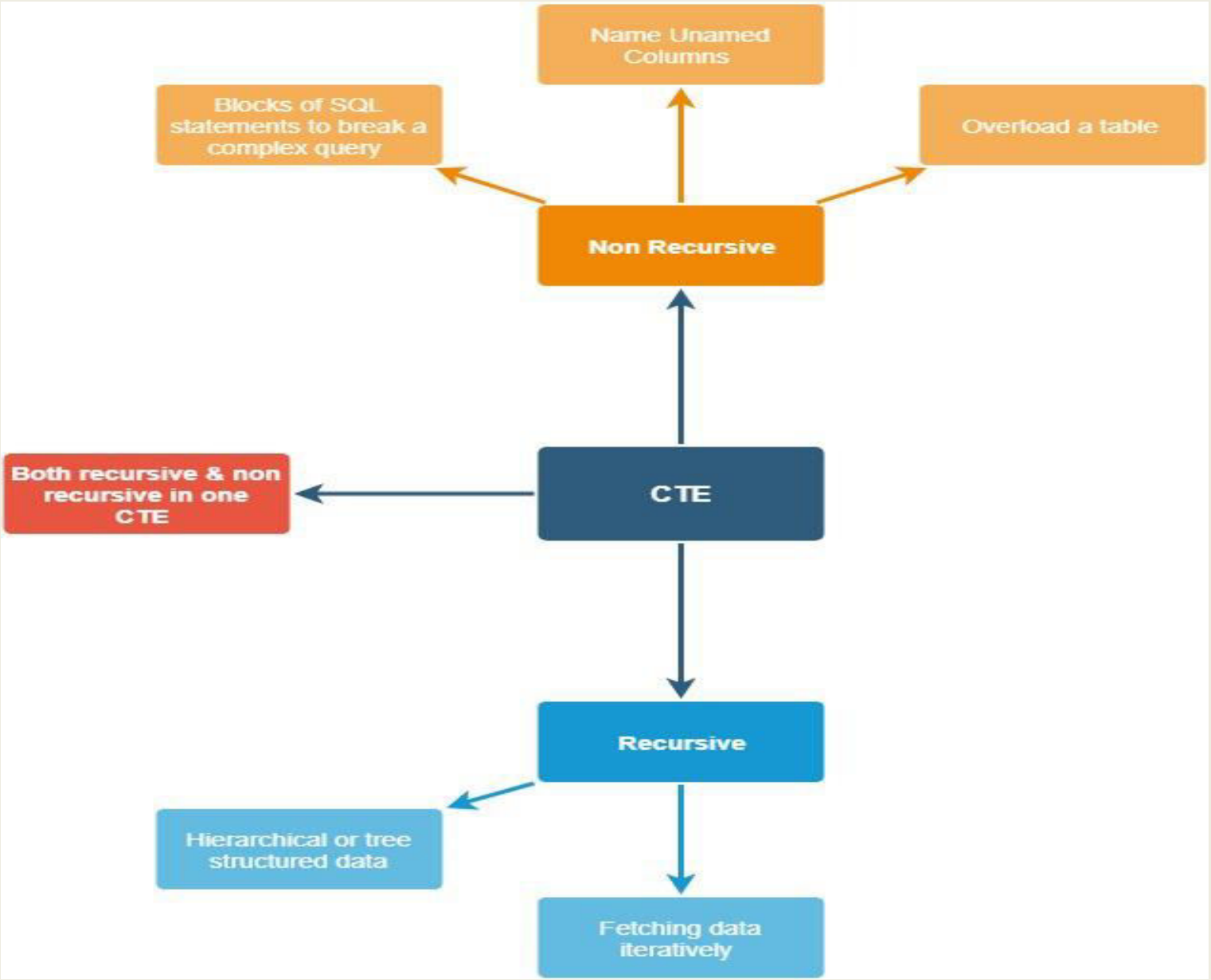
Recursive CTE -Example

-Semester wise toppers for a particular batch

-First with picks the profile of the toppers for semester 1 and is recursive over the semesters upto final

```
-WITH recursive toppers_eachsem as (select roll,name, semno,course, cgpa,
    rank()over (partition by course order by cgpa desc) as rr  from performance where
    adm_year ='2013' and semno = 1  group by roll,name, semno,course,cgpa ) UNION
    ALL
(select roll,name, a.semno,course, ,cgpa,rank()over (partition by course order by cgpa
    desc) as rr  from performance a, (select distinct semno+1 as semc from
    toppers_eachsem) where adm_year ='2013' and a.semno = c.semno  group by
    roll,name, semno,course,cgpa)
 select  course,semno, rollno,name,cgpa, rr from  toppers_eachsem where rr =1  order
    by course,semno
```

Course	Semno	Roll	Name	Cgpa	rank
AE	1	13AEXXX	Xyxyxy	9.1	1
AE	2	13AEZZZ	ZYZYZY	9.4	1


Name Unamed Columns
Blocks of SQL statements to break a complex query
Overload a table
Non Recursive
Both recursive & non recursive in one CTE
CTE
Recursive
Hierarchical or tree structured data
Fetching data iteratively

WITH CLAUSE—PERFORMANCE ISSUES IN POSTGRESQL

- While most other databases process WITH in the same way as they process views and derived tables and optimise the overall query, there is big difference in PostgreSQL.
- The PostgreSQL query planner considers each with query and the main statement separately.
- The usual query optimization methods are yet to be implemented for CTE operations
  - Projection Pushdown(eliminating unnecessary columns)
  - Predicate Pushdown (apply the predicate as early as possible)
  - Sort Elimination (redundant sort operations)
- This makes the with clause an optimization fence meta data, such as the order in which a with query returns rows, is not available during optimization of the main query.(ref: @MarkusWinad)

WITH CLAUSE—Performance Example

- explain with current_exam_session as (select subno, exam_session from exam_timetable where session = 'current' and exam_type = 'Mid') , student_exam_timetable as (select roll, a.subno, exam_session from current_exam_session a, registration b where a.subno = b.subno)

  select * from student_exam_timetable

Output

"CTE Scan on student_exam_timetable (cost=7046.72..8797.34 rows=87531 width=80)"

--In the absence of WITH

- explain select roll, a.subno, exam_session from exam_timetable a, registration b where session = 'current' and exam_type = 'Mid' and a.subno = b.subno

"Hash Join (cost=114.99..3594.84 rows=45293 width=21)"

Complex Queries –Application in Academic Information System

- Mostly Adhoc queries for checking batch operations, reports and Analytics
- Activities used include
  - Exam Scheduling (using WITH and aggregate functions) (*B Hemalatha. A Multistage Technique for Examination Timetabling. International Journal of Computer Applications 181(33):5-11, December 2018*)
  - Curriculum Compliance by generating patterns using array_agg functions
  - Class Scheduling (using WITH and array_agg)
  - Performance Ranking for various criteria (WITH in conjunction with window functions ,aggegrate functions along with the filter clause)
  - Reports needing Unpivoting/Pivoting through JSON

CONCLUSIONS

- Complex queries can be compacted through WITH clause
- The JSON and filter clause enable us to unpivot and pivot data
- Rich features available through Array_agg and statistical functions enable more compaction of queries
- Combining all the above, the data manipulation task can be delegated at the database level itself, especially for Adhoc queries

References

- https://www.postgresql.org/docs/10/index.html
- https://modern-sql.com/ --@MarkusWinand
- https://momjian.us/main/presentations/sql.html
- https://www.tutorialspoint.com/postgresql/
- http://www.postgresqltutorial.com/
- https://www.linuxjournal.com/content/postgresql-10-great-new-version-great-database

ACKNOWLDGEMENTS

- IIT Kharagpur
- The entire student community of IIT KGP
- Organizers of PGConf India 2019.