UNIT 4: STRUCTURED QUERY LANGUAGE

CONTENTS

- Data Definitions
- Basic Query Structure
- Modification of the Database
- Set operations
- Aggregate functions
- Null Values
- Nested sub queries
- Complex Queries
- Views
- Joined Relations

Data Definition Language

Allows the specification of not only a set of relations but also information about each relation, including:

- The schema for each relation, including attribute types
- Domain of values associated with each attribute
- Integrity constraints
- Security and authorizations information for each relation
- Physical storage structure of each relation on disk

Create Table Construct

- An SQL relation is defined using the create table command:

create table r ($A_1 D_1$, $A_2 D_2$, ... $A_n D_n$,
(integrity-constraints$_1$)
...,
(integrity-constraints$_k$))

- r is the name of the relation
- each A_i is an attribute name in the schema of relation r
- D_i is the data type of attribute A_i

Example:

```
create table branch
( branch_name    char (15),
  branch_city    char (30),
  assets         number (20));
```

- INTEGRITY CONSTRAINTS ON TABLES
  - not null
  - primary key

Example: Declare branch_name as primary key for branch

```
create table branch
( branch_name    char(15) primary key,
  branch_city    char(30) notnull,
  assets         number(30),
  );
```

primary key declaration on an attribute automatically ensures not null

Drop and Alter Table Constructs

- The drop table command deletes also all information about the dropped relation from their database
- The alter table command is used to add attributes to an existing relation:

  alter table r add A D

  where A is the name of the attribute to be added to relation r and D is the domain of A
  - All attributes tuples in the relation are assigned null as the value for the new attribute
- The alter table command can also be used to drop attributes of a table

  alter table r drop A

  where A is the name of an attribute of relation r

Basic Insertion and Deletion of Tuples

- Newly created table is empty
- Add a new tuple to account

```
insert into account
    values ('A-9732','Perryridge',1200)
```

- Insertion fails if any integrity constraint is violated

- Delete all tuples from account

```
delete from account
```

- Modification of the Database - Insertion
- Add a new tuple to account

```
insert into account
    values ('A-9732','Perryridge',1200)
```

or equivalently

```
insert into account (branch_name, balance, account_number)
    values ('Perryridge', 1200, 'A-9732')
```

Add a new tuple to account with Balance set to null

```
insert into account
    values ('A-777','Perryridge',null)
```

Modification of the Database - Updates

- Increase all accounts with balance over $10,000 by 6%, all other accounts receive 5%.

- Write two update statements

```
update account
 set   balance = balance * 1.06
where  balance > 10000
```

```
update account
 set   balance = balance * 1.05
where  balance <= 10000
```

Basic Query Structure

- A typical SQL query has the form:

```
select  A1, A2, ---- An
from    r1, r2, ---- rn
where   P
```

- Ai represent a attribute
- Ri represent a relation
- P is a predicate

- The query is equivalent to the relational algebra expression
- Result of an SQL query is a relation

SELECT Clause

- The select clause list the attributes desired in the result of a query
  - corresponds to project operation of the relational algebra

Example:
Find the names of all the branches in loan relation:

```
select branch_name
from   loan
```

In relational algebra

$$\Pi_{branch\_name}(loan)$$

NOTE: SQL names are case insensitive (i.e you may use upper or lower-case letters)

Eg:- Branch_name = BRANCH_NAME = branch_name

- SQL allows duplicates in relations as well as in query results
- To force the eliminations of duplicates, insert the keyword distinct after select
- Find the names of all branches in loan relations, and remove duplicates

```
select distinct branch_name
from loan
```

The keyword all specifies that duplicates not be removed

```
select all branch_name
from loan
```

- An asterisk in the select clause denotes "all attributes"

```
select *
from loan
```

- The select clause can contain arithmetic expression involving the operation +, -, * and /, and operating on constants or attributes of tuples

```
select loan_number, branch_name, amount * 100
from loan
```

WHERE Clause

- The where clause specified conditions that the result must satisfy
  - corresponds to the selection predicate of relational algebra

- To find all loan numbers for loans made at the Perryridge branch with loan amounts greater than $1200

```
select loan_number
from loan
where branch_name = 'Perryridge' and
      amount > 1200
```

- Comparisons results can be combined using the logical connectives and, or & not

- SQL includes a between comparison operator

Find the loan number of those loans with loan amounts between $90000 and $100,000

```
select loan_number
from loan
where amount between 90000 and 100000
```

FROM Clause

- The from clause lists the relations involved in the query
  - Corresponds to the cartesian product operation of the relational algebra
- Find the cartesian product borrower X loan

```
select *
from borrower, loan
```

- Find the name, loan_number and loan amount of all customers having a loan at the Perryridge branch

```
select customer_name, borrower.loan_number
       amount
from borrower, loan
where borrower.loan_number = loan.loan-
      number and branch_name='
      Perryridge'
```

RENAME OPERATION

- SQL allows renaming relations and attributes using the as clause:

  old-name as new-name

Find the name, loan_number and loan amount of all customers ;
rename the column name loan_number as loan_id

```
                                        amount
select customer_name, borrower.loan_number as loan_id,
from borrower, loan
where borrower.ban_number = loan.loan_number
```

TUPLE VARIABLES

Tuple variables are defined in the **form** clause via the use of the **as** clause

Find the customer names and their loan numbers and amount for all customers having a loan at some branch

```
select customer_name, T.loan_number, S.amount
from borrower as T, loan as S
where T.loan_number = S.loan_number
```

- Find the names of all branches that have greater assets than some branch located in Brooklyn

```
select distinct T.branch_name
from branch as T, branch as S
where T.assets > S.assets and S.branch_city =
                                    'Brooklyn'
```

Keyword as is optional and may be omitted

borrower as T = borrower T

STRING OPERATION

- SQL includes a string-matching operator for comparison on character strings. The operator 'like' uses patterns that are described using 2 special characters

  - percent (%) - % character matches any substring
  - underscore (_) - _ character matches any character

Find the names of all customers whose street includes the substring "Main"

```
select customer_name
from customer
where customer_street like '%Main%'
```

match the name "Main%"

```
like 'Main\%' escape '\'
```

SQL supports variety of string operation

- Concatenation (using ||)
- Converting from upper to lower case (using vice versa)
- finding string length

Ordering the display of Tuples

List in alphabetic order the names of all customers having a loan in Perryridge branch

```
select distinct customer_name
from borrower, loan
where borrower.loan_number = loan.loan_number and
      branch_name = 'Perryridge'
order by customer_name
```

- We may specify desc for descending order or asc for ascending order, for each attribute; ascending order is the default

order by customer_name desc

SET OPERATIONS

- The set operations union, intersect and except operate on relations and correspond to the relational algebra operations $\cup$, $\cap$, $-$

- Each of the above operations automatically eliminates duplicates; to retain all duplicates use the corresponding multiset versions union all, intersect all and except all

- Suppose a tuple occurs m times in r and n times in s then, it occurs:

  - $m+n$ times in r union all s
  - $\min(m, n)$ times in r intersect all s
  - $\max(0, m-n)$ times in r except all s

classmate
Date
Page 75

- Find all customers who have a loan, an account or both

```
select customer_name from depositor
union
select customer_name from borrower
```

- Find all customers who have both a loan and an account

```
select customer_name from depositor
intersect
select customer_name from borrower
```

- Find all customer who have an account but no loan

```
select customer_name from depositor
except
select customer_name from borrower
```

AGGREGATE FUNCTIONS

- These functions operate on the multiset of values of column of a relation and return a value

  avg: average value
  min: minimum value
  max: maximum value
  sum: sum of values
  count: number of values

- Find the average account balance at the Perryridge branch

```
select avg (balance)
from account
where branch_name = 'Perryridge'
```

- Find the number of tuples in customer relation

```
select count (*)
from customer
```

- Find the number of depositor in bank

```
select count (distinct customer_name)
from customer
```

Aggregate functions - Group by

- Find the number of employee in each dept

```
select   count(ename), deptno
from     emp
group by deptno
```

Note: Attributes in select clause outside of aggregate functions must appear in group by list

Aggregate functions - Having clause

- Find the names of all departments whose average salary is greater than 2000

```
select   count(ename), deptno
from     emp
group by deptno
having   avg(salary) > 2000
```

Note: predicated in the having clause are applied after the formation of groups whereas predicates in the where clause are applied before forming groups

Joined Relations

- Join operations takes two relations and return as a result another relation

These additional operations are typically used as subquery expression in the form clause

- Join condition - defines which tuples in the two relations match, and what attributes are present in the result of the join
- Join type - defines how tuples in each relation that do not match any tuple in the other relation (based on the join condition) are treated

Join types	Join conditions
inner join	natural
left outer join	on <predicate>
right outer join	using $(A_1, A_2, \dots, A_n)$
full outer join	

Examples

1) inner join - non-common tuples are removed but common attributes of both relation are there in result relation (i.e attribute appears twice in result)
loan inner join borrower on
loan.loan_number = borrower.loan_number

2) left outer join -

loan left outer join borrower on
loan.loan_number = borrower.loan_number

3) loan natural inner join borrower

4) loan natural right outer join borrower

- Natural join can get into trouble if 2 relations have an attribute with same name that should not affect the join condition

Solution :-

loan full outer join borrower using (loan_number)

NESTED SUBQUERIES

- SQL provides a mechanism for the nesting of subqueries
- A subquery is a select-from-where expression that is nested within another query
- A subquery is a query within a query
- Outer query is called as main query and inner query which is written in main query (where clause) is called subquery
- Single row operation (<, >, <=, >=, =, <>)
- Multiple row operation (IN, ALL, ANY)

SINGLE ROW SUBQUERY

- A subquery is one which returns only one row to main query

Find all employees whose salary is less than Rahul's salary

→
```
select ename, salary
from employee
where salary < (select Salary
                from employee
                where ENAME = 'Rahul')
```

Multiple row Subquery

- If subquery returns more than one row then that type of query is known as multiple row subquery

- IN

Find all employees having salary not equal to any of the manager's salary not be manager

```
- select emp_id, last_name, job_id, salary
  from employee
  where salary NOT IN (select salary
                       from employee
                       where job_id = 'MAN')

  And job_id <> 'MAN'
```

ANY

```
select emp_id, last_name, job_id, salary
from employee
where salary > ANY (select salary
                    from employee
                    where job_id = 'MAN')
```

ALL

- select emp_id, last_name, job_id, salary
from employee
where salary > ALL (select salary
from employee
where job_id = 'MAN')

EXSIST CLAUSE

- The existence checks whether a subquery returns any values

find employee details if employee 141 is present in employee table

select * from emp
where exsits (select eid
from emp
where eid = 141

VIEW Definition

- A relation that is not of the conceptual model but is made visible to user as a "virtual relation" is called a view

- A view is defined using the create view statement which has the form

create view V as <query expression>

where <query expression is any legal SQL expression. View name is represented by V

once a view is defined, the view name can be used to refer the virtual relation that the view generates.