STRUCTURED QUERY LANGUAGE

CONTENTS

- Data Definition
- Basic Query Structure
- Modification of the Database
- Set Operations
- Aggregate Functions
- Null Values
- Nested Sub queries
- Complex Queries
- Views
- Joined Relations**

DATA DEFINITION LANGUAGE

- The schema for each relation, including attribute types.
- Integrity constraints
- Authorization information for each relation.
- Non-standard SQL extensions also allow specification of
  - *The set of indices to be maintained for each relations.*
  - *The 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-constraint$_1$),
  ...,
  (integrity-constraint$_k$))

  - *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 the primary key for *branch*

- **create table** *branch*
  (*branch_name* char(15) **primary key ,**
  branch_city char(30) **not null**,
  assets number(30),
  );
- **primary key** declaration on an attribute automatically ensures **not null**

Drop and Alter Table Constructs

- The **drop table** command deletes all information about the dropped relation from the 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 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 relation:

  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*

  Note: Will see later how to delete selected tuples

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 balances 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
```

Account_number	Branch_number	Balance
A-101	Downtown	500
A-102	Perryidge	400
A-201	Brighton	900
A-215	Mianus	700
A-217	Brighton	750
A-222	Redwood	700
A-305	RoundHill	350

ENO	ENAME	JOB	MGR	SAL	COMM	DEPTNO
5369	SMITA	CLERK	7566	8000	0	20
5399	VASANT	SALESMAN	7566	1600	300	20
5499	ALLENA	SALESMAN	7698	1600	300	30
5521	VIKRAM	SALESMAN	7698	1250	500	30
5566	JOHINA	MANAGER	7839	5975	500	20
5698	BHAWNA	MANAGER	7839	9850	1400	30
5611	SUMIT	HOD	7839	3000		10
5839	SHEENA	CEO	9900			10
5368	ISHA	SUPERVISOR	7366	8000	0	20
5599	ROHIT	SALESMAN	7698	1600	300	30

DNO	DNAME	DLOC
10	MANAGEMENT	MAIN BLOCK
20	DEVELOPMENT	MANUFACTURING UNIT
30	MAINTAINANCE	MAIN BLOCK
40	TRANSPORT	ADMIN BLOCK
50	SALES	HEAD OFFICE

Q1. Insert the following into department and employee table.

Q2. Alter the employee table to set the default commission of all employees to Rs10000/- who are working as managers

Q3. Delete only those who are working as supervisors.

Q4. Delete the rows whose eno is 5599.

Basic Query Structure

- A typical SQL query has the form:

  select $A_1, A_2, \ldots, A_n$
  from $r_1, r_2, \ldots, r_m$
  where P

  - A_i *represents an attribute*
  - R_i *represents a relation*
  - *P is a predicate.*

- This query is equivalent to the relational algebra expression.

- The 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 the projection operation of the relational algebra*
- Example: find the names of all branches in the *loan* relation:

  select *branch_name*
  from *loan*

- In the relational algebra, the query would be:

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

- NOTE: SQL names are case insensitive (i.e., you may use upper- or lower-case letters.)
  - *E.g. Branch_Name ≡ BRANCH_NAME ≡ branch_name*

The select Clause (Cont.)

- SQL allows duplicates in relations as well as in query results.
- To force the elimination of duplicates, insert the keyword **distinct** after select.
- Find the names of all branches in the *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*

The select Clause (Cont.)

- An asterisk in the select clause denotes “all attributes”

```
select *
from loan
```

- The **select** clause can contain arithmetic expressions involving the operation, +, –, ∗, and /, and operating on constants or attributes of tuples.
- E.g.:

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

WHERE Clause

- The **where** clause specifies conditions that the result must satisfy
  - *Corresponds to the selection predicate of the relational algebra.*

- To find all loan number 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

- Comparison results can be combined using the logical connectives **and, or,** and **not.**

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'
```

customer_name	*loan_number*
Adams	L-16
Curry	L-93
Hayes	L-15
Jackson	L-14
Jones	L-17
Smith	L-11
Smith	L-23
Williams	L-17

BORROWER

Loan_Number	Branch_Name	Amount
L-11	ROUND HILL	900
L-14	DOWNTOWN	1500
L-15	PERRYRIDGE	1500
L-16	PERRYRIDGE	1300
L-17	DOWNTOWN	1000
L-23	REDWOOD	2000
L-93	MIANUS	500

LOAN

RENAME Operation

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

  old-name **as** *new-name*

- E.g. Find the name, loan number and loan amount of all customers; rename the column name *loan_number* as *loan_id.*

- **select** *customer_name, borrower.loan_number* **as** *loan_id, amount*
  from *borrower, loan*
  where *borrower.loan_number = loan.loan_number*

Tuple Variables

- Tuple variables are defined in the **from** 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*

Loan_Number	Branch_Name	Amount
L-11	ROUND HILL	900
L-14	DOWNTOWN	1500
L-15	PERRYRIDGE	1500
L-16	PERRYRIDGE	1300
L-17	DOWNTOWN	1000
L-23	REDWOOD	2000
L-93	MIANUS	500

LOAN

BRANCH_NAME	BRANCH_CITY	ASSESSTS
BRIGHTON	BROOKLYN	7100000
DOWNTOWN	BROOKLYN	9000000
MIANUS	HORSENECK	40000
NORTHTOWN	RYE	3700000
PERRYRIDGE	HORSENECK	1700000
POWNAL	BENNINGTON	30000
REDWOOD	PALO ALTO	2100000
ROUNDHILL	HORSENECK	8000000

BRANCH

customer_name	*loan_number*
Adams	L-16
Curry	L-93
Hayes	L-15
Jackson	L-14
Jones	L-17
Smith	L-11
Smith	L-23
Williams	L-17

BORROWER

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.
  - *Example:* ***order by*** *customer_name* ***desc***

STRING OPERATION

- SQL includes a string-matching operator for comparisons on character strings. The operator “like” uses patterns that are described using two special characters:
  - *percent (%). The % character matches any substring.*
  - *underscore (_). The _ 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 a variety of string operations such as
  - *concatenation*
  - *converting from upper to lower case (and vice versa)*
  - *finding string length*

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*

Set Operations

- 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 as well as an account.

  (**select** *customer_name* **from** *depositor*)
  intersect
  (select *customer_name* **from** *borrower)*

- Find all customers 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 a 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

Aggregate Functions (Cont.)

- Find the average account balance at the Perryridge branch.

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

- Find the number of tuples in the *customer* relation.

  select count (*)
  from *customer*

- Find the number of depositors in the bank.

  select count (distinct *customer_name)*
  from *depositor*

Aggregate Functions – Group By

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT		11/17/1981	5000		10
7698	BLAKE	MANAGER	7839	5/1/1981	2850		30
7782	CLARK	MANAGER	7839	6/9/1981	2450		10
7566	JONES	MANAGER	7839	4/2/1981	2975		20
7788	SCOTT	ANALYST	7566	12/9/1982	3000		20
7902	FORD	ANALYST	7566	12/3/1981	3000		20
7369	SMITH	CLERK	7902	12/17/1980	800		20
7499	ALLEN	SALESMAN	7698	2/20/1981	1600	300	30
7521	WARD	SALESMAN	7698	2/22/1981	1250	500	30
7654	MARTIN	SALESMAN	7698	9/28/1981	1250	1400	30
7844	TURNER	SALESMAN	7698	9/8/1981	1500	0	30
7876	ADAMS	CLERK	7788	1/12/1983	1100		20
7900	JAMES	CLERK	7698	12/3/1981	950		30
7934	MILLER	CLERK	7782	1/23/1982	1300		10

- Find the number of employees in each department.

  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(sal)>2000
```

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

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT		11/17/1981	5000		10
7698	BLAKE	MANAGER	7839	5/1/1981	2850		30
7782	CLARK	MANAGER	7839	6/9/1981	2450		10
7566	JONES	MANAGER	7839	4/2/1981	2975		20
7788	SCOTT	ANALYST	7566	12/9/1982	3000		20
7902	FORD	ANALYST	7566	12/3/1981	3000		20
7369	SMITH	CLERK	7902	12/17/1980	800		20
7499	ALLEN	SALESMAN	7698	2/20/1981	1600	300	30
7521	WARD	SALESMAN	7698	2/22/1981	1250	500	30
7654	MARTIN	SALESMAN	7698	9/28/1981	1250	1400	30
7844	TURNER	SALESMAN	7698	9/8/1981	1500	0	30
7876	ADAMS	CLERK	7788	1/12/1983	1100		20
7900	JAMES	CLERK	7698	12/3/1981	950		30
7934	MILLER	CLERK	7782	1/23/1982	1300		10

Joined Relations

- **Join operations** take two relations and return as a result another relation.
- These additional operations are typically used as subquery expressions in the **from** 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
inner join
left outer join
right outer join
full outer join

Join Conditions
natural
on <predicate>
using $(A_1, A_1, \ldots, A_n)$

Joined Relations – Datasets for Examples

loan_number	*branch_name*	*amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

loan

customer_name	*loan_number*
Jones	L-170
Smith	L-230
Hayes	L-155

borrower

Note: borrower information missing for L-260 and loan information missing for L-155

loan **inner join** *borrower* **on**
loan.loan_number = borrower.loan_number

loan_number	*branch_name*	*amount*	*customer_name*	*loan_number*
L-170	Downtown	3000	Jones	L-170
L-230	Redwood	4000	Smith	L-230

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

loan_number	*branch_name*	*amount*	*customer_name*	*loan_number*
L-170	Downtown	3000	Jones	L-170
L-230	Redwood	4000	Smith	L-230
L-260	Perryridge	1700	*null*	*null*

- *loan* **right outer join** *borrower* **on**
  loan.loan_number = borrower.loan_number

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-155	*null*	*null*	Hayes

Joined Relations – Examples

- Natural join can get into trouble if two relations have an attribute with same name that should not affect the join condition
  - *e.g. an attribute such as remarks may be present in many tables*
- *Solution:*
  - *loan* ***full outer join*** *borrower* ***using*** *(loan_number)*

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	*null*
L-155	*null*	*null*	Hayes

Nested Subqueries

- SQL provides a mechanism for the nesting of sub queries.
- 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.

Example: Find all the 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 he should not be manager.*

EMP_ID	LAST_NAME	JOB_ID	SALARY
124	SMITA	MANAGER	5800
141	SIMRAN	CLERK	3500
142	MOHAN	CLERK	5800
143	MARK	CLERK	2600
144	MUDIT	CLERK	2500
178	MAHENDRA	MANAGER	6800

  - *SELECT emp_id,last_name,job_id, salary*
    From employee
    Where salary NOT IN (select salary
    from employee
    where job_id='MANAGER') and job_id<>'MANAGER'

- ANY
  - *SELECT emp_id,last_name,job_id, salary*
    From employee
    Where salary > ANY (select salary
    from employee
    where job_id='MANAGER')
- ALL
  - *SELECT emp_id,last_name,job_id, salary*
    From employee
    Where salary > ALL (select salary
    from employee
    where job_id='MANAGER')

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 exists (select eid
                    from emp
                        where eid=141)
```

VIEW DEFINITION

- A relation that is not of the conceptual model but is made visible to a 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. The view name is represented by *v.*
- Once a view is defined, the view name can be used to refer to the virtual relation that the view generates.

Advantages of Views:

1) SECURITY

- View can restrict user from accessing all data.
- In case of view only data that is given in view is accessible to user. So all data of base table is not accessible to user which will give you security of information.

2) HIDES COMPLEXITY

- The view may be result of very complex query. Hence instead of writing such complicated query again and again we can store such result to a view and access it whenever we want to access.

Disadvantages of Views:

1) PERFORMANCE

- DBMS translates queries of view to queries on basic table.
- As complexity is hidden by view hence, users are not aware of how much complicated task the query is actually performing.

2) UPDATE RESTRICTION

- Update is possible for simple views and not for complex.
- Complex views are generally considered as read only type of views.

ENO	ENAME	JOB	MGR	SAL	COMM	DEPTNO
5369	SMITA	CLERK	7566	8000	0	20
5399	VASANT	SALESMAN	7566	1600	300	20
5499	ALLENA	SALESMAN	7698	1600	300	30
5521	VIKRAM	SALESMAN	7698	1250	500	30
5566	JOHINA	MANAGER	7839	5975	500	20
5698	BHAWNA	MANAGER	7839	9850	1400	30
5611	SUMIT	HOD	7839	3000		10
5839	SHEENA	CEO	9900			10
5368	ISHA	SUPERVISOR	7366	8000	0	20
5599	ROHIT	SALESMAN	7698	1600	300	30

DNO	DNAME	DLOC
10	MANAGEMENT	MAIN BLOCK
20	DEVELOPMENT	MANUFACTURING UNIT
30	MAINTAINANCE	MAIN BLOCK
40	TRANSPORT	ADMIN BLOCK
50	SALES	HEAD OFFICE

- List the records in the empl table orderby salary in ascending order.
- List the records in the empl table orderby salary in descending order.
- Display only those employees whose dno is 30.
- Display dno from the table employee avoiding the duplicated values.
- Create a manager table from the empl table which should hold details of only about the managers.
- List the employee names and the department name in which they are working.
- List the records in sorted order of their employees.
- List the jobs whose average salary is greater than 2000.

Account_number	Branch_number	Balance
A-101	Downtown	500
A-102	Perryidge	400
A-201	Brighton	900
A-215	Mianus	700
A-217	Brighton	750
A-222	Redwood	700
A-305	RoundHill	350

ACCOUNT

Loan_Number	Branch_Name	Amount
L-11	ROUND HILL	900
L-14	DOWNTOWN	1500
L-15	PERRYRIDGE	1500
L-16	PERRYRIDGE	1300
L-17	DOWNTOWN	1000
L-23	REDWOOD	2000
L-93	MIANUS	500

LOAN

BRANCH_NAME	BRANCH_CITY	ASSESSTS
BRIGHTON	BROOKLYN	7100000
DOWNTOWN	BROOKLYN	9000000
MIANUS	HORSENECK	40000
NORTHTOWN	RYE	3700000
PERRYRIDGE	HORSENECK	1700000
POWNAL	BENNINGTON	30000
REDWOOD	PALO ALTO	2100000
ROUNDHILL	HORSENECK	8000000

BRANCH

CUSTOMER_NAME	ACCOUNT_NUMBER
HAYES	A-102
JOHNSON	A-101
JOHNSON	A-201
JONES	A-217
LINDSAY	A-222
SMITH	A-215
TURNER	A-305

DEPOSITOR

customer_name	*loan_number*
Adams	L-16
Curry	L-93
Hayes	L-15
Jackson	L-14
Jones	L-17
Smith	L-11
Smith	L-23
Williams	L-17

BORROWER