Unit-2

Relational Algebra

* relational algebra ← consists of a basic set of operations for the relational model

* relational algebra operations

UNARY

(i) SELECT : σ
(ii) PROJECT : π
(iii) RENAME : ρ

RELATIONAL ALGEBRA

(i) UNION : ∪
(ii) INTERSECTION : ∩
(iii) DIFFERENCE : −
(iv) CARTESIAN PRODUCT

JOINS ⋈

LEFT OUTER JOIN ⟕
RIGHT OUTER JOIN ⟖

AGGREGATE FUNCTIONS = ℱ

Company Database Schema

Employee

Fname	Minit	Lname	SSN	Bdate	Address	Sex	Salary	Super-ssn	Dno

Department

Dname	Dnumber	mgr-ssn	mgr-start-date

Dept_locations

Dnumber	Dlocation

Project

Pname	Pnumber	Plocation	Dnum

works_on

essn	pno	hours

dependent

essn	dependent_name	sex	bdate	relationship

Questions

(1) Select the employee tuples whose department number is 4

$\sigma_{DNO=4}(EMPLOYEE)$

(2) Select the employee tuples whose salary is greater than 30,000

$\sigma_{salary>30000}(EMPLOYEE)$

(3) Select the employee tuples whose salary is greater than 30,000 and whose deptno is 4.

$\sigma_{DNO=4\ AND\ SALARY>3000}(EMPLOYEE)$

(4) Select the employee tuples whose department number is 4 or those whose salary is greater than 30,000

$\sigma_{DNO=4\ OR\ SALARY>30,000}(EMPLOYEE)$

(5) Select the employee's first & last name and salary

$\pi_{FNAME, LNAME, SALARY}(EMPLOYEE)$

(6) Retrieve the firstname, last name & salary of all employees who work in dept. no 5.

$DEP5\_EMPS \leftarrow \sigma_{DNO=5}(EMPLOYEE)$

$RESULT \leftarrow \pi_{FNAME, LNAME, SALARY}(DEP5\_EMPS)$

(7) Retrieve the social security nos. of all employees who either work in department 5 or directly supervise an employee who works in department 5.

$$DEP5\_EMPS \leftarrow \sigma_{DNO=5}(EMPLOYEE)$$

$$RESULT1 \leftarrow \pi_{SSN}(DEP5\_EMPS)$$

$$RESULT2 \leftarrow \pi_{SUPERSSN}(DEP5\_EMPS)(SSN)$$

$$RESULT \leftarrow RESULT1 \cup RESULT2$$

(8) Retrieve the name of the manager of each department

$$DEPT\_MGR \leftarrow DEPARTMENT \bowtie_{SSN=MGR\_SSN} EMPLOYEE$$

$$RESULT \leftarrow \pi_{FNAME, LNAME}(DEPT\_MGR)$$

(9) List all the employee names and the name of the departments they manage if they happen to manage a dept

$$DEPT\_MANAGE \leftarrow EMPLOYEE \bowtie_{SSN=SUPER\_SSN} DEPARTMENT$$

$$RESULT \leftarrow \pi_{FNAM, LNAM, DNAME}(DEPT\_MANAGE)$$

(10) Retrieve the deptno, the no. of employees in each dept, and their average salary.

$$RESULT \leftarrow {}_{DNO}\mathcal{F}_{COUNT\ SSN,\ AVERAGE\ SALARY}(EMPLOYEE)$$

grouping attr.

(11) Retrieve the name and address of all employee who work for the 'Research' department

$$RESEARCH\_ROWS \leftarrow \sigma_{dname = 'Research'}(DEPARTMENT)$$

$$RESEARCH\_EMPS \leftarrow RESEARCH\_ROWS \bowtie_{DNO = Dnumber} EMPLOYEE$$

$$RESULT \leftarrow \pi_{Fname, Lname, Address}(RESEARCH\_EMPS)$$

(12) For every project located in Stafford, list the project no, the controlling department, the department manager lastname, address and birth date.

$$STAFFORD\_PROJS \leftarrow \sigma_{Plocation = 'Stafford'}(PROJECT)$$

$$STAFFORD\_PROJ\_DEPTS \leftarrow STAFFORD\_PROJS \bowtie_{DNUM = Dnumber} DEPARTMENT$$

$$STAFFORD\_PROJ\_MANAGERS \leftarrow STAFFORD\_PROJ\_DEPTS \bowtie_{Ssn = mgr\_ssn} Employee$$

$$RESULT \leftarrow \pi_{pno, dname, lname, address, bdate}(STAFFORD\_PROJ\_MANAGERS)$$

(13) List the names of all employees w/ 2 or more dependents.

$T_1 \leftarrow {}_{ESSN}\gamma_{COUNT\ dependent\_name}(DEPENDENT)$

(no-of-dependents)

$T_2 \leftarrow \sigma_{no\_of\_dependents > 2}(T_1)$

$RESULT \leftarrow \pi_{LNAME, FNAME}(T_2 * EMPLOYEE)$

(14) Retrieve the names of employees with no dependents

$ALL \leftarrow \pi_{SSN}(EMPLOYEE)$

$DEP \leftarrow \pi_{SSN}(DEPENDENT)$

~~$RESULT \leftarrow \pi_{FNAME, LNAME}(ALL -$~~

$EMPS\_W/O\_DEPS \leftarrow (ALL - W/O DEP)$

$DETAILS \leftarrow EMPS\_W/O\_DEPS * EMPLOYEE$

$RESULT \leftarrow \pi_{FNAME, LNAME}(DETAILS)$

(15) Make a list of project numbers for projects that involve an employee whose last name is 'Smith' either as a worker or as a manager of the department.

(i) get ssn of all Smiths

$SMITHS \leftarrow \pi_{ssn}(\sigma_{LNAME = 'Smith'}(EMPLOYEE))$

(ii) Find the works-on

$SMITH\_WORKSON \leftarrow \pi_{pno}(\ ?\ WORKS\_FOR * SMITHS)$ ①

(iii) Get all managers

$MGRS \leftarrow \pi_{LNAME, DNUMBER} (EMPLOYEE \bowtie_{SSN = MGR\text{-}SSN} DEPARTMENT)$

(iv) Smith managed departments

$SMITH\_MG\_DEPT \leftarrow \pi_{DNUM} (\sigma_{LNAME = 'smith'} (MGRS))$

$SMITH\_MG\_PROJECT \leftarrow \pi_{pnum} (SMITH\_MG\_DEPT\_Project)$

↳ (2)

Result ← (1) ∪ (2)

(16) List the names of managers who have at least one dependent.

$MGRS \leftarrow \pi_{mgr\text{-}ssn} (DEPARTMENT)$

$EMPS\_W|\_DEP \leftarrow \pi_{essn} (DEPT)$

$MGR\_W|\_DEPS \leftarrow MGRS \cap EMPS\_W|DEP$

$RESULT \leftarrow \pi_{FNAME, LNAME} (MGR\_W\_DEPS * EMPLOYEE)$

Additional Exercises

* Consider the following schema.

LIB_BRANCH

Branchid	Branch_name	address

BORROWER

cardno	name	address	phone

BOOK_LOANS

Bookid	Branchid	cardno	Date-out	Due-date

(1) List the names of all borrowers who do not have any books checked out.

(i) get names of all borrowers

$$ALL\_BORROWERS \leftarrow \Pi_{name}(BORROWER)$$

(ii) join BORROWER and BOOK_LOANS

$$BOOK\_BORROWERS \leftarrow \Pi_{name}(BOOK\_LOANS \bowtie_{cardno = cardno} BORROWER)$$

(iii) Find the difference.

$$RESULT \leftarrow ALL\_BORROWERS - BOOK\_BORROWERS$$

(2) Retriere the names, address and no. of books checked out for all borrowers who have more than 5 books checked out

$$COUNT\_BOOKS \leftarrow {}_{cardno}\gamma_{count\ bookid.}(BOOK\_LOANS)$$
(cardno, book_num)

$$MORE\_THAN\_FIVE \leftarrow \sigma_{book\_num > 5}(COUNT\_BOOKS)$$

$$MORETHAN\_FIVE\_DETAILS \leftarrow MORE\_THAN\_FIVE \bowtie_{cardno = cardno} BORROWER$$

$$RESULT \leftarrow \Pi_{name, address, book\_num}(MORE\_THAN\_FIVE\_DETAILS)$$

* Consider the following schema.

HOUSESALES

Address	City	Datesold	Price

HOUSES

Address	City	No. of bedrooms	no. of bathrooms	sq ft

(1) List all the houses that were sold in 2000 w/ more bathrooms than bedrooms. List the address, city, no. of bedrooms & bathrooms

MORE_BATHROOMS $\leftarrow \sigma_{\text{no-of-batrooms} > \text{no-of-bedrooms}}$ (HOUSES)

~~MORE_BATHROOM_DETAILS $\leftarrow \Pi_{\text{Address, City}}$ (MORE-BATHROOMS)~~

2000_HOUSESALES $\leftarrow \sigma_{\text{Datesold}=2000}$ (HOUSESALES)

DETAILS $\leftarrow$ MORE-BATHROOM ⋈ 2000-HOUSESALES
address = address

RESULT $\leftarrow \Pi_{\text{address, city, no-of-bathrooms, no-of-bedrooms}}$ (DETAILS)

check

(2) List all the houses that were sold and then later sold again for less money. Include the address, city & both dates sold & both prices

(i) create 2 copies of the housesales table

$HS1 \leftarrow \Pi_{address, city, datesold, price}(housesales)$

$HS2 \leftarrow \Pi_{address, city, datesold, price}(housesales)$

(ii) join both tables on the condition that the datesold is not the same

$new\_table \leftarrow HS1 \bowtie HS_2$

(oldprice, olddate)

newprice, newdate

$hs1.address = hs_2.address$

$hs1.city = hs2.city$

$hs1.datesold \neq hs_2.datesold$

(iii) join housesales and newtable on the condition that the datesold in house-sales > ~~new~~ new-table and

house-sales.price > new-table.price

HOUSES_SOLD_AGAIN ← house sales ⋈ new-tables

house-sales.price < new-tables.price

AND

house-sales.datesold > new-tables.date

(iv) RESULT ← $\Pi_{address, city, oldprice, olddate, newprice, newdate}$ (HOUSES_SOLD_AGAIN)

Unit 2 - Relational Model and SQL

* Data Abstraction: Refers to the suppression of details of data organization and storage

→ highlights the essential features

→ DB supports data abstraction, so that diff. users perceive data at their preferred level of detail.

* Data Model : a collection of concepts to describe

a. Structure of database

b. Basic operations for manipulating these structures

c. Valid user defined operations that are allowed on the

* Data model Structure & Constraints

→ constructs used to define the database structure

→ constructs include elements as well as groups of elements and relationships among such groups

→ constraints specify some restrictions on valid data

* Data model operations - used for specifying database retrievals & updates

* Categories of data models -

(i) Conceptual (High-level, semantic) data models

entity based data models

have entities : represent real world objects or concepts

attributes : some property of interest that further describes the entity

relationships : an association between the entities

(ii) Object based data model - extending the E-R model w/ notions of encapsulation, functions & object identity.

(iii) Physical (low-level, internal) data models

→ details how data is stored in the computer

→ specified in an ad-hoc manner through DBMS design

(ii) Implementation (representational) data models

→ uses a collection of tables to represent both data & relationships among data.

→ Table has columns & each column has a unique name

(v) Self-describing data models

→ model combines the description of the data w/ the data values themselves

* Three-Schema Architecture

A. The internal level

→ describes the physical storage structure of the database

→ uses physical data model

B. Conceptual level

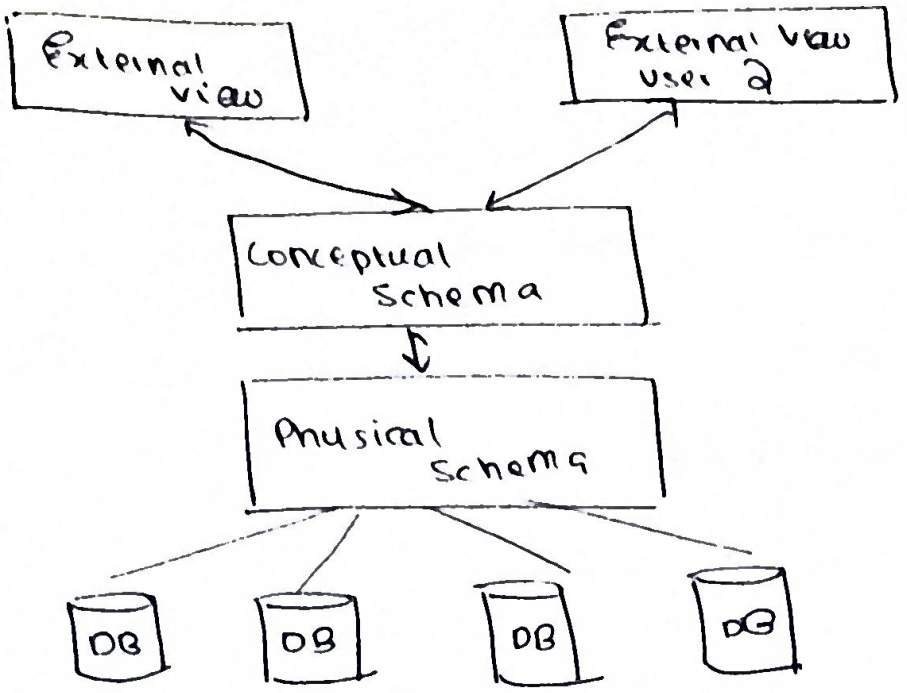
→ describes the structure of the db, hides the details of the physical storage

→ uses conceptual data model

C. External level

→ describes the part of the database that a particular user group is interested in

→ implemented using representational data model.



* Schema : → description of the database
→ displays the name of record types & some constraints

* Database State or Instance : actual data stored in a database at a particular moment in time
→ schema changes infrequently, database state changes every time the database is updated.

* Relational Model → based on the idea of sets
→ has a set of rows, each row is called a tuple
→ each column has a column header called the attribute name.

* Formal Definitions

① Relational Schema : denoted by $R(A_1, A_2 \dots A_n)$

R = relation

$A_1, A_2 \dots A_n$ = attributes

each attribute has a domain D in the relational schema R.

② Domain → a set of atomic values
→ has a logical definition and also has a data type or format defined for it.

eg. Customer (CustId, Custname, Address, Phone)
domain of CustId is 6 digit nos.

(3) Tuple : ordered set of values enclosed in < >
(each row of the table)

(4) Relation State : Subset of the cartesian product of the domain of its attributes

$$r(R) \in dom(A_1) * dom(A_2) \ldots dom(An)$$

↙

a specific state,
with a set of tuples

* Characteristics of Relational Model

1. Tuples in a relation are unordered
2. Attributes in a relation are ordered
3. Values in a tuple are considered atomic.

* Relational Model Constraints

A. Model-based constraints or implicit constraints

→ Constraints inherent in the data model, like how there cannot be duplicate tuples.

B. Schema-based constraints or explicit constraints

→ constraints directly expressed in the schema of the data model, specified using DDR statements

C. Application-based or semantic constraints

→ Constraints enforced by application programs
eg. triggers & views

* Domain Constraints

→ domain constraints specify that within each tuple, the value of each attribute A must be an atomic value from the domain

→ domain constraint violation takes place if it does not obey datatype constraints.

* Keys : 3 types

(i) candidate keys

(ii) primary & alternate keys

(iii) foreign keys

A. Candidate Keys:

→ If K is the set of attributes on R, the K is the candidate key if it has the following properties:

(i) uniqueness : no value of 2 distinct tuples of R contains the same value of K

(ii) minimal superkey : A superkey from which we cannot remove any attributes, and still have the uniqueness constraint hold

eg.

ID	First	Last
S139	John	Smith
S140	Mary	Jones
S141	John	Brown
S142	Jane	Smith

ID = unique

(ID, First), (ID, last), (ID, First, Last) satisfy uniqueness but are not minimal

candidate keys are ID and (First, Last)

Superkeys: No 2 tuples in any valid relation instance have the same value for the set of attributes.

$$t_1[SK] \neq t_2[SK]$$

Any such subsets are called superkeys.

B. Primary and Alternate Keys

→ A given relation may have 2 or more candidate keys. Exactly one of those keys is chosen as the primary key, the others are then called alternate keys.

C. Foreign Key

→ used to specify a relation among tuples in two relations: the referencing relation and the referenced relation.

→ A foreign key is a set of attributes, say FK of the referencing relation whose values are required to match values of the candidate key CK of the referenced relation.

* Entity Integrity

→ No primary key (PK) can be NULL, in any tuple of r(R). This is because primary key values are used to uniquely identify the individual tuples in a relation

* Referential Integrity: The database must not contain any unmatched foreign key values.

Example

Employee

ID	Name	DID
15	John Smith	13
16	Mary Jones	14
17	John Braon	13
18	Jane Smith	Null

Dept. Table

DID	DName
13	Marketing
14	Accounts
15	Personnel

Each employee's DID value is either NULL, or it matches an entry in the dept. relation

* Referential Integrity Violations

→ When relations are updated, referential integrity can be violated.

→ To compensate, the options are:

(i) cascade: actions cascaded to the matching tuples

(ii) restrict: restricted to the case where there are no matching tuples

(iii) nullify - make values null

Example:

DID	DName
13	Marketing
14	Accounts
15	Personnel

ID	Name	DID
15	John Smith	13
16	Mary Jones	14
17	John Brown	13
18	Jane smith	Null

Case1: if marketing's DID is changed to 16, with cascade, John Smith's and John Brown's DID changes

Case2: If accounts is deleted, then so is Mary Jones

Case3: If Marketing's DID changes or accounts is deleted, u0| NULLIFY John Smith, John Brown and Mary Jones' DIDs become NULL.

* DDR Commands

(1) Create Table

```
CREATE TABLE <tablename>
 (   <colname>   <datatype + length>,
     :
 );
```

② Constraints

→ To enforce rules at the table level

→ Prevent the deletion of a table if there are dependencies

The following constraints exist:

(i) NOT NULL
(ii) UNIQUE
(iii) PRIMARY KEY
(iv) FOREIGN KEY
(v) CHECK

→ Constraints can be at the

(i) column level – references a single column

(ii) table level constraint – references one or more columns & is defined separately from column definitions

② CREATE TABLE w/ constraints

(i) column level

```
CREATE TABLE <table name>
( column name <datatype - length> CONSTRAINT
    <constraint name> <constraint type>,
    :
    :
)
```

(ii) table level

```
CREATE TABLE <table name>
(
    : column definitions
    :

CONSTRAINT <constraint name> <constraint type> (column, ...)
);
```

(3) Foreign key in create table statements

(i) column level

CREATE TABLE <table name> (

<columnname> <datatype (length)> CONSTRAINT <constraint name> REFERENCES <other table name> (column from 2nd table)

:

);

(ii) table level

CREATE TABLE <table name> (

<column defn>

:

CONSTRAINT <constraint-name> FOREIGN KEY

<column names- - - - >

REFERENCES <other table name> (column list . . .) .);

Options (i) ON DELETE CASCADE

(ii) ON DELETE set NULL

* Check & Unique constraints are implemented the same way.

* Alter Table Commands

(1) To add columns

ALTER TABLE <table name>

ADD (<columname> <datatype - length>);

(2) To modify columns

ALTER TABLE <table name>

MODIFY (<columname datatype (newlength)) ,

(3) To drop columns

ALTER TABLE <tablename>

DROP COLUMN <column name>;

(4) To add a constraint

ALTER TABLE <table name>

ADD CONSTRAINT <constraint-name>

<constraint type> ;

eg. ALTER TABLE employees

ADD CONSTRAINT emp-manager-fk

FOREIGN KEY (manager-id)

REFERENCES employees (employee-id) ;

(5) To drop a constraint.

ALTER TABLE <table name>

DROP CONSTRAINT < constraint name>;

(6) Enabling and disabling constraints

Enabling constraints

ALTER TABLE <table name>

enable CONSTRAINT <constraint name>;

Disable constraint

ALTER TABLE <table name>

disable CONSTRAINT <constraint name> ;

* Truncating Tables

→ used to delete all the rows in a table, cannot use rollback after this

TRUNCATE TABLE <table name>

* Drop Table

→ all data and structure in the table is deleted

→ Any pending transactions are committed

→ cannot roll back after this

DROP TABLE <table name>;

* Data Manipulation Language

DML commands include: (i) SELECT
(ii) INSERT
(iii) UPDATE
(iv) DELETE

* Basic DML Query Structure

Has 3 basic clauses:

SELECT $A_1, A_2 \ldots A_n$
FROM $R_1, R_2 \ldots R_m$
WHERE P

A = attributes
R = relation
P = predicate identifying which tuples have to be retrieved

* Query Examples

Consider the schema of the company database

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super Ssn	Dno

(1) To select all attributes

→ SELECT * FROM employee;

(2) To select just the first and last name

→ SELECT Fname, Lname
FROM employee;

(3) To select only unique salary values

→ SELECT distinct (salary)
FROM employee;

(4) Retrieve the birthdate and address of the employee whose name is John Smith

→ SELECT Bdate, Address
FROM employee
WHERE fname = 'John' AND lname = 'Smith';

(5) Retrieve the lname, sex, salary of the male employees whose salary is greater than 30,000;

SELECT lname, sex, salary
FROM employees
WHERE Sex = 'M';
AND salary > 30000';

(6) Retrieve the details of the employee whose salary is greater than 50,000 or should belong to deptno 5.

→ SELECT * FROM employee
WHERE Salary > 50,000 OR dno = 5;

(7) Retrieve the details of employees who do not belong to deptno. 5.

→ SELECT * FROM employee
WHERE dno != 5;

(8) Add 300 to all the salaries

```
SELECT fname, salary, salary+300 AS new_salary
FROM employee;
```

(9) Compute the annual salary of all the employees

```
SELECT fname || lname AS Name, 12 * salary AS
annual_salary
FROM employee;
```

(10) Select those employees who have not been allotted with a manager

```
SELECT fname, lname
FROM employee
WHERE super_ssn IS NULL;
```

(11) Retrieve all employee names whose first name starts with 'J'

```
SELECT fname, lname
FROM employee
WHERE fname LIKE 'J%';
```

(12) SELECT names of all employees who were born during the 1950s

```
SELECT fname, lname
FROM employee
WHERE Bdate LIKE '%5_';
```

(13) Retrieve all employees in dept 5 whose salary is between 30,000 and 40,000.

```
SELECT * FROM employee
WHERE (salary BETWEEN 30000 AND 40000)
AND dno = 5;
```

(14) Display the details of all employees whose superssn is 5 or 4 or 1.

→ SELECT * FROM employee
WHERE dno IN (5, 4, 1);

(15) SELECT the employees based on their birthdate

→ SELECT lname, fname, bdate
FROM employee
ORDER BY bdate, (ASC / DESC)

(16) Select all the employees sorted by their annual salaries

SELECT fname, lname, salary * 12 AS annual-salary
FROM employee
ORDER BY annual-salary

(17) Sort all the employees by their dept no & in decreasing order of salary

SELECT * FROM employee
ORDER BY deptno, salary DESC;

* SQL Functions

Types:
- Single Row Functions — Operate on single rows & return one result per row
- Multiple Row Functions — Manipulate groups of rows to give one result per group of rows

* Case Manipulation Function

(i) LOWER (<col>)
(ii) UPPER (<col>)
(iii) INITCAP (<col>)
(iv) CONCAT (<col1>, <col2>)
(v) SUBSTR (<col>, <no. of pos>)
(vi) LENGTH (<col>)
(vii) INSTR (<col>, 'str')

(viii) LPAD (<col> , n , 'string')

(ix) RPAD (<col> , n, 'string'))

(x) TRIM (leading | trailing | both)

(xi) REPLACE (text , search string, replacement string)

Consider the schema for the employee table.

employee_id	first_name	last_name	email	phone_number	hire_date	job_id	salary	comm_pct	mgr_id	dept_id

(1) Select the first & last names of all the employees with the first letter of their name capitalized.

```
SELECT  initcap(first_name) First_Name,
initcap(last_name) Last_Name
FROM employee;
```

(2) Display the first 5 characters of the employees' last name.

```
SELECT  last_name , substr(last_name 1, 5 )
                                  from 1st
FROM employees;                   char, 5pos
```

(3) Display the last character of the employees' last name

```
SELECT  last_name , substr (last_name , -1)
FROM employee;
```

(4) Display the last name, and also the length of the last name of the employees.

```
SELECT last_name, length (last_name)
FROM employees;
```

★★★

(5) Display the index of the second occurrence of the character 'a' in the last name

```
SELECT last_name, instr(lastname, 'a', 1, 2)
```

↓ char to search for ↓ pos to start from → which occurrence?

(6) Make the total length of the salary 20, by padding * on both sides

```
SELECT rpad(salary, 10, '*'), lpad(salary, 10, '*')
FROM employee;
```

(7) Cut off the word 'He' from 'HelloWorld'

```
SELECT TRIM('He', 'HelloWorld') FROM dual;
```

* Math Functions

round(<no>, no. of places)

trunc(<no>, no. of places)

ceil(<num>)

floor(<num>)

(8) Round, Truncate, Ceil and Floor the no 43.235

Round to 2 digits

Truncate to 1 digit

```
SELECT round(43.235, 2),
       ceil(43.235)
       floor(43.235),
       trunc(43.235)
FROM dual;
```

* Date Functions

MONTHS - BETWEEN

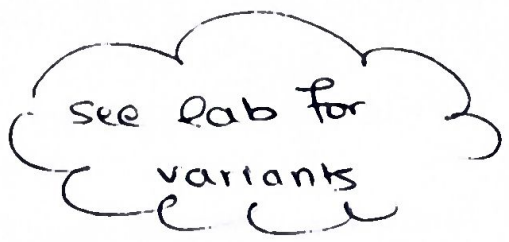
ADD - MONTHS — add a specific no of months to calendar date

NEXT DAY - next day of the date specified

LAST - DAY - last day of the month

ROUND - round date

TRUNC - truncate date



(9) SELECT the current date

→ SELECT sysdate FROM dual;

(10) Find out the date 5 days ago.

→ SELECT sysdate - 5 FROM dual;

(11) Find the hire date and the no. of months between the hire employed.

→ SELECT hiredate, MONTHS_BETWEEN (sysdate, hire_date)
AS months
FROM employee;

(12) Display the last names, and the no. of weeks between the hire date and the sysdate.

SELECT ename, ROUND(sysdate - hire_date) / 7 as WEEKS
FROM employee;

(13) Display the date of the next Friday

SELECT NEXT_DAY (sysdate, 'Friday') FROM dual;

(14) Display the tenure of the employee in months as well as the date of review which is 6 months after the hire date

→ SELECT last_name, hire_date, MONTHS_BETWEEN
(sysdate, hire_date) AS tenure,
ADD_MONTHS(hire_date, 6) AS review_date
FROM employee;

(15) Date Formatting

ddspth mon yyyy = Thirteenth Apr 2023

ddsp month yyyy = Thirteen April 2023

ddsp month yyyysp = Thirteen April ~~Twenty~~ Two Thousand Twenty Three

* Aggregate Functions

min(), max(), sum(), avg()

(16) Display the average, maximum, minimum & ~~sum~~ total salary of the employees

→ SELECT avg(salary) average,
max(salary) max_sal,
min(salary) min_sal,
sum(salary) total_sal
FROM employee;

(17) Display the avg. salary, maximum & min salary of employes whose job id is SA_REP,

→ SELECT avg(salary), max(salary), min(salary)
FROM employee
WHERE job_id LIKE 'SA_REP';

(18) Display the minimum & maximum hire date of the employees

→ SELECT min(hire_date), max(hire_date)
FROM employees;

* Group By Clause

→ used to group rows by any column, and return summary results
→ must use HAVING to restrict groups

(19) Display the average salaries of employees based on each department.

```
SELECT avg(salary)
FROM employee
GROUP BY dept-id;
```

(20) Display the avg, max & min salary of employees based on the job id

```
SELECT job_id, avg(salary), max(salary), min(salary)
FROM employee
GROUP BY job_id;
```

(21) Display the sum(salary) of the employees based on the job_id within the department

```
SELECT dept_id, job_id, SUM(salary)
FROM employees
GROUP BY job_id, dept_id
```

(22) Display the total salary and their dept no. of employees having manager w/ manager id 100

```
SELECT dept-id, manager-id, sum(salary)
FROM employee
GROUP BY dept-id
HAVING manager-id = 100;
```

(23) List the no. of employees in each dept.

```
SELECT count(employee-id)
FROM employee
GROUP BY dept-id
```

(24) List the no. of employees under each manager of the departments

```
SELECT count(employee-id)
FROM employee
GROUP BY department-id, manager-id
```

(25) List the departments whose average salary is greater than 6000

```
SELECT department-id, avg(salary)
GROUP BY department-id
HAVING avg(salary) > 6000;
```

* Joins

Cartesian Product : all rows in the first table are joined to all rows in the second table

→ To avoid a cartesian product, include a valid join condition in a where clause.

* To join n tables, need n-1 join conditions

Consider the schemas of 3 tables

Employees

emp-id	last-name	dept-id

Departments

dept-id	d-name	located

Locations

Location-id	City

A. Equi-join

→ Joins based on the = operator

→ Equi join removes null values

(1) Display the details of employees along with their department name.

```
SELECT e.last_name, d.dept_id, d.d_name
FROM employees e, departments d
WHERE e.dept_id = d.dept_id;
```

(2) Display the location name of the dept where the employees work

```
SELECT e.last_name, e.dept_id, d.dept_name, l.city
FROM employees e, departments d, locations l
WHERE e.dept_id = d.dept_id
AND d.location_id = l.location_id;
```

Consider the following 2 schemas.

Employees

last_name	salary

Job_grades

grade	lowest_sal	Highest_sal

(3) Retrieve the grade of the employees

```
SELECT e.last_name, j.grade
FROM employees e, job_grades j
WHERE e.salary BETWEEN
j.lowest_sal AND j.highest_sal;
```

* Outer Joins

→ If a row does not satisfy a join condition, that row will not appear in the result.

→ Outer joins show the rows that do not meet the join condn.

→ The outer join operator is the plus sign (+) and it is placed on the side of the join that is deficient in information (no matching row,)

Consider the schemas

Employees

Empid	last_name	dept_id

Departments

dept_id	d_name	location_id

(4) List the department id, department name of all departments with or w/o employees.

```
SELECT d.dept_id, d.d_name
FROM employees e, departments d
WHERE e.dept_id (+) = d.dept_id
```

In the employee table, some emps may not have dept-id. The NULL values are present in this table. Use the (+) sign on this table.

* Self Joins

→ Joining a table to itself is self-join, searching the same table twice

Consider this schema:

employee

emp-id	last_name	mgr-id

(5) Select the name of each employee's manager

```
SELECT   emp.last_name , mgr. last_name
FROM   employee emp, employee mgr
WHERE   emp.mgr-id = mgr. emp_id
```

* Join Types

(i) CROSS JOIN – returns a cartesian product

(ii) NATURAL JOIN – joins 2 tables based on the same column name

(iii) USING (<col names>): performs an equijoin based on the column name

(iv) ON (condition): performs an equijoin based on the condn. in the ON clause

(v) LEFT | RIGHT | FULL OUTER join Table 2 – Retrieve both matching & unmatching tuples

(6) Write a query to match the dept. name w/ its location

Departments

dept-id	d-name	location-id

locations

location-id	city

```
SELECT * FROM
departments JOIN location
USING (location_id)
```

(or)

```
SELECT * FROM
departments d JOIN locations l
on (d. location_id = l. location_
                        id);
```

* Left, Right and Full Outer Joins

Employee

emp_id	last_name	dept_id

Departments

dept_id	d_name	location_id

(7) Display all employees w/ or w/o depts

→ SELECT e.last_name, d.d_name

FROM employee e,

LEFT OUTER JOIN department d.

ON (e.dept_id = d.dept_id) } USING (dept_id)

(8) Display all the departments w/ or w/o employees

SELECT e.last_name, d.d_name

FROM employee e

RIGHT OUTER JOIN department d

on (e.dept_id = d.dept_id);

(9) Display all employees & depts regardless of whether employees have depts or depts have employees.

SELECT e.last_name, d.d_name

FROM employees e

FULL OUTER JOIN departments d

on (e.dept_id = d.dept_id);

* Sub Queries

→ The inner query or subquery returns a value that is used by the outer query or the main query.

→ A subquery is a select statement that is within a clause of another SELECT statement

→ Subqueries are useful when we need to select rows from a table w/ a condition that depends on the data in the table itself

→ Nested queries can be written with single-row operators or multiple row operators (IN, ANY, ALL). (>, =, <, <>, <=)

Consider the following schemas.

Employees

~~Employees~~ ssn	last_name	first_name	salary	d_id

Department

dept_id	d_name

=ALL not really used (since no value = all the values in the subquery)

>ALL = >(max)

<ALL = <(min)

① Which employees have salaries greater than John's salary?

```
SELECT * FROM
employee
 WHERE salary >
  ( SELECT salary
    FROM employee
    WHERE first_name = 'John');
```

② Display the employee name whose dept_id is the same as employee 141.

```
SELECT last_name FROM employees
WHERE d_id = ( SELECT dept_id FROM employee
               WHERE ssn = 141);
```

(3) Display employees whose d_id is the same as that of employee 141 and whose salary is greater than that of employee 143.

```
SELECT last_name FROM employee
WHERE d_id =
(SELECT d_id FROM employee WHERE SSN = 141)
AND salary>
(SELECT salary FROM employee WHERE SSN =143);
```

(4) Display the employee_last name, d_id and salary of all employees whose salary is a minimum among all employees.

```
SELECT * FROM employee
WHERE salary = (SELECT min(salary) FROM employee);
```

(5) Display all the departments that have a minimum salary greater than department 50.

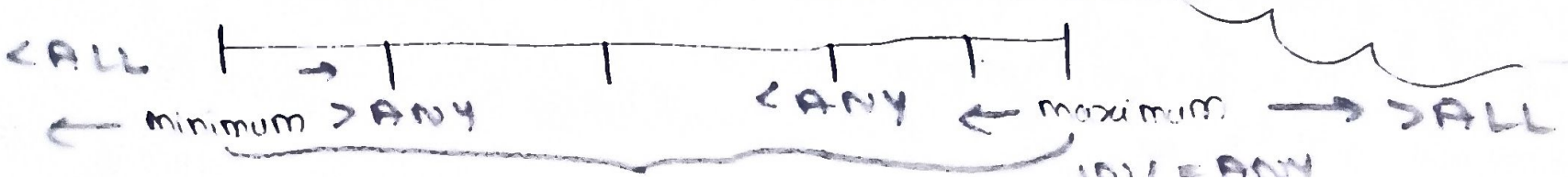
```
SELECT d_id, min salary FROM employee
GROUP BY d_id HAVING min(salary)
(SELECT salary FROM employee
 WHERE d_id =50);
```

(6) Find the employees who earn the same salary as the minimum salary for each department.

=ANY & IN are the same

>ANY = >min(value)

<ANY = <min(value)



```
SELECT f_name FROM employee
WHERE salary IN
        (SELECT min(salary) FROM employee
         GROUP BY d_id);
```

(7) Find the employees whose salary is greater than or equal to the maximum salary of any of the depts.

```
SELECT f_name FROM employee
WHERE salary >= ANY
        (SELECT max(salary) FROM
         employee
         GROUP BY d_id);
```

(8) Find the employees whose salary is greater than or equal to the maximum salary of all the departments.

```
SELECT f_name FROM employee
WHERE salary >= ALL
        (SELECT max(salary) FROM employee
         GROUP BY d_id);
```

* Correlated subQueries
  - → used for row by row processing
  - → each subquery is executed once for every row of the outer query
* Nested Sub Queries vs. Correlated SubQueries

Nested Sub Queries → The inner query executes first and finds a value.

→ The outer query executes once, using the value from the inner query.

Correlated subquery Execution

→ Get a candidate row - fetched by the outer query

→ Execute the inner query using the value of the candidate row.

→ Use values resulting from the inner query to qualify or disqualify the candidate.

→ Repeat until no candidate row remains.

* General Syntax

```
SELECT , column1 , column2 ...
FROM   table1  outer
WHERE  column1  operator
        (SELECT  column1, column2
         FROM table2
         where  expr1 =
                        outer.expr2);
```

(1) Find all employees who earn more than the average salary in their department.

```
SELECT * FROM employee outer
WHERE ( SELECT avg(salary) FROM employee
        WHERE d_id = outer.d_id );
```

(2) Retrieve the name of the employees who has a dependendent with the same first name and is the same sex as the employee

```
SELECT  E.L.name
FROM employee E   WHERE  E.employee_id  IN
(SELECT d.employee_id FROM dependent
 WHERE  e.L_name = d.dependent_name
AND  e.sex = d.sex);
```

* The EXISTS operator

→ Used to test if a value retrieved by the outer query exists in the results of the values retrieved by the inner query.

→ If the subquery returns at least one row, the operator returns TRUE, otherwise it returns FALSE.

① List the dept. names which have at least one employee.

```
SELECT d.d_name FROM dept d
WHERE exists
        (SELECT * FROM employee WHERE e.d_id = d.d_id);
```

② Retriero the name of the employee who has a dependent w/ the same first name and is the same sex as the employee.

```
SELECT  e.L_name FROM employee e
WHERE exists
( SELECT d.emp_id FROM dependent d
   WHERE  e.emp_id = d.emp_id
   AND  e.f_name = d.dependent_name
   AND  e.sex = d.sex);
```

(3) List all the dept. names which do not have any employees

```
SELECT  d.d_name FROM departments d

WHERE  NOT EXISTS

( SELECT * FROM employee e

 WHERE  d.d_id = e.d_id );
```

(4) List the name of the employees who do not have any dependents

```
SELECT  e.lname  FROM  employees e

WHERE  NOT EXISTS

( SELECT * FROM dependents d

  WHERE  e. emp_id  =  d.emp_id );
```

(5) List the name of managers who have at least one dependent

```
SELECT  e.lname  FROM employees e

WHERE  exists

( SELECT  *  FROM  department d

 WHERE  e. emp-id  =  d. mgr_id)

AND EXSTS

(SELECT * FROM dependents dp

WHERE  e. emp-id  = d. emp-id );
```

Unit - 2

Relational Model and SQL

Additional Queries

Table Structure

salesman

salesman_id	name	city	commission

customer

customer_id	customer_name	city	grade	salesman_id

order

order_no	purch_amt	order_date	customer_id	salesman_id

(1) Find the name & city of those customers & salesmen who live in the same city

```
SELECT   c.customer_name, c.city
FROM     customer c,
JOIN     salesman s
 ON      s.salesman_id = c.salesman_id
WHERE    s.city = c.city;
```

(2) Find the names of all the customers along with the salesman who works for them

```
SELECT  c.customer_name , s.salesman_name
FROM    customer c
JOIN    salesman s
ON      c.salesman_id = s.salesman_id;
```

(3) Display orders by the customers not located in the same cities where the salesmen live.

```
SELECT  order_no
FROM    orders o
JOIN    customer c
ON      o.customer_id = c.customer_id
JOIN    salesman c
ON      s.salesman_id = c.salesman_id
WHERE   s.city != c.city;
```

(4) Use a subquery to display all the orders issued by the salesman 'Paul'

```
SELECT * FROM orders
WHERE salesman_id =
(SELECT salesman_id FROM
 salesman WHERE name = 'Paul');
```

(5) Using a subquery, display all the orders where the value are greater than the average order value for 10th Oct. 2012

```
SELECT * FROM orders
WHERE   purch_amt >
        (SELECT avg(purch_amt)
         FROM orders
         WHERE ord_date = '10-OCT-2012');
```

(6) Using a subquery, find all the orders attributed to salesmen in Paris

```
SELECT * FROM orders
WHERE salesman_id IN (SELECT salesman_id FROM
                      Salesman WHERE city = 'Paris');
```

(7) Using a subquery, extract the order details for the salesman who earned the maximum commission

```
SELECT * FROM orders
WHERE salesman_id
 IN
(SELECT salesman_id FROM salesman WHERE commission=
(SELECT max(commission) FROM salesman));
```

(8) Find the name and ids of all salesman who had more than one customer.

```
SELECT  s.salesman_id ,s.salesman_name, COUNT(c.salesman_id)

FROM  salesman s

JOIN  customer c  ON

s.sailsman_id = c.customer_id

GROUP BY  s.salesman_id , s.salesman_name

HAVING count (c.salesman_id) > 1;
```

(9) Write a query to find all the salesman who worked for 1 customer.

do like previous c.salesman_id = 1

(10) Display all the orders that had amounts that were greater than at least one of their orders from Sep 10, 2012.

```
SELECT * FROM orders

WHERE purch_amt >ANY

( SELECT  purch_amt FROM orders

   WHERE  ord_date = '10-SEP-2012');
```

→ Consider the student registration database consisting of the following schema

Student

sno	name	address	phone

Course

cno	description	hours	pno

Professor

pno	name	office

registration

sno	cno	date

① What are the courses taught by a specific professors 'P001' and 'P002' ?

```
SELECT  c.cno, p.no
FROM    course  c
JOIN    professor  p  ON
P.pno = c.pno
WHERE   P.pno  IN ('P001', 'P002');
```

(2) Who teaches the course CS 6302 and where is his/her office?

```
SELECT p.name, p.office
FROM professor p
 JOIN course c  ON
p.pno = c.pno
WHERE  c.cno = 'CS 6302';
```

(3) Who are the professors handling courses registered by the student 'S102'?

```
SELECT p.name  FROM
professor p
JOIN course c  ON     c.pno  =p.pno
JOIN registration r  ON  r.cno =  c.cno
 WHERE  r.sno = 'S102';
```

(4) List out courses not registered by a student (use exists)

S104

```
SELECT  c.cno
FROM  course  c
WHERE NOT EXISTS (
SELECT  r.cno FROM  registration r
    WHERE  r.sno ='S104');
```

* Consider the following relational schema

Flights

flno	from	to	distance	departs	price

Aircraft

aid	aname	cruisingrange

Certified

eid	aid

employees

eid	ename	salary

Note that the employees relation describes pilots and other kinds of employees as well. Every pilot is certified for some aircraft, & only pilots are certified to fly.

(1) For each pilot who is certified for more than 3 aircraft, find the eid and the maximum cruising range of the aircraft that he/she is certified for.

```
SELECT c.eid , a.cruisingrange
FROM   aircraft a
JOIN   certified c
ON     a.id = c.eid
WHERE  c.eid IN (
                 SELECT  c.eid , COUNT (c.aid) AS
                  aircraft-count
                 FROM    certified c
                 GROUP BY c.eid
                 HAVING  aireraft-count >3 );
```

(2) Find the names of pilots whose salary is less than the price of the cheapest route from Los Angeles to Honolulu.

```
SELECT  e.ename, e.salary
FROM    employees e
WHERE   e.salary < ( SELECT min(price) P.
                     FROM flights
                     WHERE from = 'Los Angeles'
                     AND to = 'Honolulu');
```

??? no table join?

(3) Find the aircraft ids of all the aircraft that can be used on routes from Los Angeles to Chicago.

```
SELECT  a.aid
FROM    aircraft a, flights
WHERE   a.cruising range < flights.distance;
```

* Consider the following schema.

Suppliers

sid	sname	address

Parts

pid	pname	color

Catalog

sid	pid	cost

a. Find the sids of suppliers who supply some red or green part.

```
SELECT c.sid , p.color
FROM catalog c
JOIN parts P
ON c.pid = p.pid
WHERE p.color IN ('red', 'green');
```

b. Find pairs of sids such that the supplier with the first sid charges more for some part than the supplier with the second sid.

```
SELECT distinct c1.sid AS first_sid, c2.sid AS second_sid
FROM catalog c1
JOIN catalog c2
ON c1.pid = c2.pid
WHERE c1.sid <> c2.sid
AND c1.cost > c2.cost;
```

c. Find the sids of suppliers who supply some red part or are at 221 Packer Sheet.

```
SELECT s.sid
FROM suppliers s
JOIN catalog c
ON s.sid = c.sid
JOIN parts p
ON p.pid = c.pid
WHERE
```