DBMS

UNIT-3

SQL

feedback/corrections: vibha@pesu.pes.edu

VIBHA MASTI

SQL

- Declarative language
- SQL : core components + extensions (eg: GIS)

DDL

1. CREATE

- Schemas, tables, domains, views, assertions, triggers
- PSQL : database

1.1 Create Schema

```
CREATE SCHEMA COMPANY AUTHORIZATION 'Jsmith';
```

- Catalog: named collection of schemas
  - Schema called INFORMATION_SCHEMA
- Default owner: root (postgres in PSQL)
  Default schema (creating schema): owner name

1.2 Create Table

- Default schema (creating table): PUBLIC

```
CREATE TABLE EMPLOYEE
```
← PUBLIC schema

- Prefix table name with Schema name while creating

```
CREATE TABLE COMPANY.EMPLOYEE
```
(COMPANY: schema)

- Alternate: create schema name auth name (

. . .

. . .

create table statements

)

1.3 Virtual Relations

```
CREATE VIEW NAME AS (QUERY);
```

Column_name attribute_type constraint,

T1

```
CREATE TABLE EMPLOYEE
        ( Fname                 VARCHAR(15)             NOT NULL,
          Minit                 CHAR,
          Lname                 VARCHAR(15)             NOT NULL,
          Ssn                   CHAR(9)                 NOT NULL,
          Bdate                 DATE,
          Address               VARCHAR(30),
          Sex                   CHAR,
          Salary                DECIMAL(10,2),
          Super_ssn             CHAR(9),
          Dno                   INT                     NOT NULL,
        PRIMARY KEY (Ssn),
CREATE TABLE DEPARTMENT
        ( Dname                 VARCHAR(15)             NOT NULL,
          Dnumber               INT                     NOT NULL,
          Mgr_ssn               CHAR(9)                 NOT NULL,
          Mgr_start_date        DATE,
        PRIMARY KEY (Dnumber),
        UNIQUE (Dname),
        FOREIGN KEY (Mgr_ssn) REFERENCES EMPLOYEE(Ssn) );
CREATE TABLE DEPT_LOCATIONS
        ( Dnumber               INT                     NOT NULL,
          Dlocation             VARCHAR(15)             NOT NULL,
        PRIMARY KEY (Dnumber, Dlocation),
        FOREIGN KEY (Dnumber) REFERENCES DEPARTMENT(Dnumber) );
CREATE TABLE PROJECT
        ( Pname                 VARCHAR(15)             NOT NULL,
          Pnumber               INT                     NOT NULL,
          Plocation             VARCHAR(15),
          Dnum                  INT                     NOT NULL,
        PRIMARY KEY (Pnumber),
        UNIQUE (Pname),
        FOREIGN KEY (Dnum) REFERENCES DEPARTMENT(Dnumber) );
CREATE TABLE WORKS_ON
        ( Essn                  CHAR(9)                 NOT NULL,
          Pno                   INT                     NOT NULL,
          Hours                 DECIMAL(3,1)            NOT NULL,
        PRIMARY KEY (Essn, Pno),
        FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn),
        FOREIGN KEY (Pno) REFERENCES PROJECT(Pnumber) );
CREATE TABLE DEPENDENT
        ( Essn                  CHAR(9)                 NOT NULL,
          Dependent_name        VARCHAR(15)             NOT NULL,
          Sex                   CHAR,
          Bdate                 DATE,
          Relationship          VARCHAR(8),
        PRIMARY KEY (Essn, Dependent_name),
        FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn) );
```

Figure 6.1 SQL CREATE TABLE data definition statements for defining the COMPANY schema from Figure 5.7.

Attribute Types

(a) String

- Fixed length: CHAR(n), CHARACTER(n)
- Variable length: VARCHAR(n), CHAR VARYING(n), CHARACTER VARYING(n)
- Enclose in '' single quotes
- Default n = 1
- Fixed length: blank-padded

(b) Numeric

- Integer: INTEGER, INT, SMALLINT
- Float: FLOAT, REAL, DOUBLE PRECISION
- Decimal: DECIMAL(i,j), DEC(i,j), NUMERIC(i,j)
  - i: precision (total digits)
  - j: scale (no. of decimal places - defaut = 0)
  - DECIMAL(5,2), DEC(10)

(c) Bit String

- Fixed length: BIT(n), Variable length: BIT VARYING(n)
- Default n = 1
- Enclosed in '' and prefixed with B (eg: B'1001')

(d) Boolean

- True and False (and unknown for 3-valued logic)

(e) Date

- Ten positions: YYYY-MM-DD
- Enclosed in single quotes ' '
- Sometimes prefixed with DATE
- Eg: DATE '2019-02-28'

(f) Time

- Eight positions: HH:MM:SS
- Time(i) — i extra digits for decimal of seconds

additional

(g) Timestamp

- DATE + TIME + 6 positions (fraction of second)
- Optional timezone qualifier
- TIMESTAMP '2021-09-28 19:40:30.643792'

(h) Interval

- Relative value to increment / decrement date, time, timestamp types

Large Object Types

(a) CLOB

- Character large object (text documents)
- PSQL: text (1 GB max)
- Oracle: max 4 GB
- Eg: `book review clob(10KB)`

(b) BLOB

- Binary Large Objects
- Images, videos etc
- PSQL: bytea (max 1GB)
- Oracle: max 4 GB
- Eg. `image blob(10MB)`
  `movie blob(2GB)`
- PSQL: use psycop2 to insert images

1.4 Create Domain

- Renamed default datatype with or without constraints
- Easy to change datatype for domain
- Improves readability
- Eg: `CREATE DOMAIN SSN_TYPE AS CHAR(9);`

1.5 Create Type

- User Defined Types (UDTs)
- Object-oriented apps (not RDBMS)
- PSQL: relational object DBs

```
CREATE TYPE full_address AS (city VARCHAR(90), street VARCHAR(90));
```

Constraints in SQL

1. Attribute Constraints

1.1 NOT NULL

- Implicit for PK

1.2 DEFAULT

- DEFAULT <value>

```
vibhamasti=# CREATE TABLE STUDENT (
vibhamasti(# SID INTEGER PRIMARY KEY,
vibhamasti(# SNAME VARCHAR(2) CHECK (SNAME NOT LIKE '% %'),
vibhamasti(# CRID INTEGER,
vibhamasti(# AGE INTEGER CHECK (AGE > 20 AND AGE < 35),
vibhamasti(# DID INTEGER,
vibhamasti(# CNO INTEGER,
vibhamasti(# COLLEGE_NAME VARCHAR(20) DEFAULT 'PESU',
vibhamasti(# CONSTRAINT FK_CRID FOREIGN KEY(CRID) REFERENCES STUDENT(SID)
vibhamasti(# );
CREATE TABLE
```

```
vibhamasti=# CREATE TABLE DEPT (
vibhamasti(# DID INTEGER PRIMARY KEY,
vibhamasti(# DNAME VARCHAR(20) NOT NULL UNIQUE
vibhamasti(# );
CREATE TABLE
```

```
vibhamasti=# CREATE TABLE COURSE (
vibhamasti(# CID INTEGER PRIMARY KEY,
vibhamasti(# CNAME VARCHAR(20)
vibhamasti(# );
CREATE TABLE
```

```
vibhamasti=# ALTER TABLE STUDENT
ADD CONSTRAINT FK_DID FOREIGN KEY(DID) REFERENCES DEPT(DID) ON DELETE CASCADE;
ALTER TABLE
```

```
vibhamasti=# ALTER TABLE STUDENT DROP CONSTRAINT FK_CRID;
ALTER TABLE
vibhamasti=# ALTER TABLE STUDENT ADD CONSTRAINT FK_CRID FOREIGN KEY(CRID) REFERE
NCES STUDENT(SID) ON DELETE SET NULL;
ALTER TABLE
```

```
vibhamasti=# ALTER TABLE STUDENT
ADD CONSTRAINT FK_CID FOREIGN KEY(CNO) REFERENCES COURSE(CID);
ALTER TABLE
```

```
vibhamasti=# ALTER TABLE STUDENT ALTER COLUMN CNO SET DEFAULT 101;
ALTER TABLE
```

```
vibhamasti=# ALTER TABLE STUDENT ADD CONSTRAINT FK_CID FOREIGN KEY(CNO) REFERENC
ES COURSE(CID) ON DELETE SET DEFAULT;
ALTER TABLE
```

2 SELECT- FROM- WHERE

SELECT <attribute list>
FROM <table list>
WHERE <condition>;

where

- <attribute list> is a list of attribute names whose values are to be retrieved by the query.
- <table list> is a list of the relation names required to process the query.
- <condition> is a conditional (Boolean) expression that identifies the tuples to be retrieved by the query.

2.2 Aliasing

```
SELECT E.Fname, S.Lname from EMPLOYEE AS E, EMPLOYEE AS S
WHERE E.SSN = S.SSN
```

2.3 Missing WHERE

- select all rows
- π relational operation
- Cartesian product - multiple tables

```
SELECT * FROM EMPLOYEE, DEPARTMENT
```

2.4 Joining

```
SELECT * FROM EMPLOYEE, DEPARTMENT JOIN DEPARTMENT ON (DNO= DNUMER);
```

- Can specify
  - LEFT OUTER JOIN
  - NATURAL JOIN
  - RIGHT OUTER JOIN

```
SELECT * FROM EMPLOYEE, DEPARTMENT NATURAL JOIN DEPARTMENT
```

2.5 Distinct Project

```
SELECT DISTINCT ESSN FROM DEPENDENT;
```

2.6 Default Project

- By default: select all

```
SELECT ALL SALARY FROM EMPLOYEE;
```

Query 0. Retrieve the birth date and address of the employee(s) whose name is 'John B. Smith'.

```
Q0:   SELECT   Bdate, Address
      FROM     EMPLOYEE
      WHERE    Fname = 'John' AND Minit = 'B' AND Lname = 'Smith';
```

Result:

Bdate	Address
1965-01-09	731Fondren, Houston, TX

2.7 Set Operations

- UNION
- EXCEPT (difference) postgres , MINUS - mysql
- INTERSECTION

2.8 Pattern Matching

- Postgres - only varchar
- Oracle - dates also

- LIKE

- _ : one char
  % : any number of chars

```
SELECT FNAME, ADDRESS FROM EMPLOYEE WHERE ADDRESS LIKE '%Houston%',
```

- For dates in postgres - typecast

```
SELECT BDATE FROM EMPLOYEE WHERE BDATE::TEXT LIKE '1965%';
```

2.9 BETWEEN

- Including

```
SELECT ... WHERE SALARY BETWEEN 10000 AND 50000;
```

2.10 Finalized Project / Arithmatic Operations

```
SELECT FNAME, SALARY, 1.1*SALARY AS HIKED FROM EMPLOYEE;
```

2.11 Ordering of Tuples

- ORDER BY
- Default: ASC
- DESC

```
SELECT LNAME FROM EMPLOYEE ORDER BY SALARY DESC;
```

- Order by multiple columns

```
SELECT * FROM EMPLOYEE, DEPARTMENT JOIN DEPARTMENT ON (DNO= DNUMER)
                                    ORDER BY DNAME, FNAME;
```

- Diff orders

```
SELECT * FROM EMPLOYEE, DEPARTMENT JOIN DEPARTMENT ON (DNO= DNUMER)
                                    ORDER BY DNAME DESC, FNAME;
```

```
SELECT * FROM EMPLOYEE, DEPARTMENT JOIN DEPARTMENT ON (DNO= DNUMER)
                                    ORDER BY DNAME DESC, FNAME;
```

3. INSERT

```
U1:  INSERT INTO  EMPLOYEE
     VALUES       ( 'Richard', 'K', 'Marini', '653298653', '1962-12-30', '98
                    Oak Forest, Katy, TX', 'M', 37000, '653298653', 4 );
```

3.1 Insert with Select

```
U3A: CREATE TABLE      WORKS_ON_INFO
     ( Emp_name        VARCHAR(15),
       Proj_name       VARCHAR(15),
       Hours_per_week  DECIMAL(3,1) );

U3B: INSERT INTO       WORKS_ON_INFO ( Emp_name, Proj_name,
                       Hours_per_week )
     SELECT            E.Lname, P.Pname, W.Hours
     FROM              PROJECT P, WORKS_ON W, EMPLOYEE E
     WHERE             P.Pnumber = W.Pno AND W.Essn = E.Ssn;
```

4. DELETE

```
U4A:  DELETE FROM  EMPLOYEE
      WHERE        Lname = 'Brown';
U4B:  DELETE FROM  EMPLOYEE
      WHERE        Ssn = '123456789';
U4C:  DELETE FROM  EMPLOYEE
      WHERE        Dno = 5;
U4D:  DELETE FROM  EMPLOYEE;
```

5. UPDATE

```
U5:   UPDATE  PROJECT
      SET     Plocation = 'Bellaire', Dnum = 5
      WHERE   Pnumber = 10;
```

ADVANCED DATATYPES — LARGE OBJECTS

- Postgres:
  1. text: large text (Oracle - clob)
  2. bytea: images (byte array) (Oracle - blob)
  3. ntext: even larger text

- psycopg2: python

COMPLEX SQL

1.1 NULL & Three-Valued Logic

- NULL due to 3 reasons

1. Unknown value. A person's date of birth is not known, so it is represented by NULL in the database. An example of the other case of unknown would be NULL for a person's home phone because it is not known whether or not the person has a home phone.

2. Unavailable or withheld value. A person has a home phone but does not want it to be listed, so it is withheld and represented as NULL in the database.

3. Not applicable attribute. An attribute LastCollegeDegree would be NULL for a person who has no college degrees because it does not apply to that person.

- Three-valued logic (not bool) — True, False, Unknown

Table 7.1 Logical Connectives in Three-Valued Logic

(a)

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

(b)

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

(c)

NOT	
TRUE	FALSE
FALSE	TRUE
UNKNOWN	UNKNOWN

Eg

Query 18. Retrieve the names of all employees who do not have supervisors.

```
Q18:    SELECT    Fname, Lname
        FROM      EMPLOYEE
        WHERE     Super_ssn IS NULL;
```

1.2 Nested Queries, Tuples and set/Multiset Comparisons

Query 4. Make a list of all project numbers for projects that involve an employee whose last name is 'Smith', either as a worker or as a manager of the department that controls the project.

```
Q4A:    ( SELECT    DISTINCT Pnumber
          FROM      PROJECT, DEPARTMENT, EMPLOYEE
          WHERE     Dnum = Dnumber AND Mgr_ssn = Ssn
                    AND     Lname = 'Smith' )
          UNION
        ( SELECT    DISTINCT Pnumber
          FROM      PROJECT, WORKS_ON, EMPLOYEE
          WHERE     Pnumber = Pno AND Essn = Ssn
                    AND     Lname = 'Smith' );
```

using nested

```
Q4A:    SELECT    DISTINCT Pnumber
        FROM      PROJECT
        WHERE     Pnumber IN
                  ( SELECT    Pnumber
                    FROM      PROJECT, DEPARTMENT, EMPLOYEE
                    WHERE     Dnum = Dnumber AND
                              Mgr_ssn = Ssn AND Lname = 'Smith' )
                  OR
                  Pnumber IN
                  ( SELECT    Pno
                    FROM      WORKS_ON, EMPLOYEE
                    WHERE     Essn = Ssn AND Lname = 'Smith' );
```

Tuple Comparisons

1.2.1 IN

... where (essn, pno) in (select essn, pno ... where);

```
SELECT    DISTINCT Essn
FROM      WORKS_ON
WHERE     (Pno, Hours) IN    ( SELECT    Pno, Hours
                              FROM      WORKS_ON
                              WHERE     Essn = '123456789' );
```

- IN operator is same as = ANY

1.2.2 ANY & ALL

- salary > all sals in dept 5

```
SELECT    Lname, Fname
FROM      EMPLOYEE
WHERE     Salary > ALL    ( SELECT    Salary
                            FROM      EMPLOYEE
                            WHERE     Dno = 5 );
```

- = all & > all may give ϕ in most instances
- = any or = some

1.2.3 ALIAS

Query 16. Retrieve the name of each employee who has a dependent with the same first name and is the same sex as the employee.

```
Q16:  SELECT    E.Fname, E.Lname
      FROM      EMPLOYEE AS E
      WHERE     E.Ssn IN    ( SELECT    D.Essn
                              FROM      DEPENDENT AS D
                              WHERE     E.Fname = D.Dependent_name
                              AND E.Sex = D.Sex );
```

- PSQL: no need to write AS (optional)

1.3 CORRELATED QUERIES

- correlated queries: for every tuple in outer query, inner query is executed (inner query uses result of outer query)
- normal: inner then outer

```
Q16A:   SELECT   E.Fname, E.Lname
        FROM     EMPLOYEE AS E, DEPENDENT AS D
        WHERE    E.Ssn = D.Essn AND E.Sex = D.Sex
                 AND E.Fname = D.Dependent_name;
```

more optimised

1.4 EXISTS and UNIQUE

- check if inner query is empty

like IN

```
Q16B:   SELECT   E.Fname, E.Lname
        FROM     EMPLOYEE AS E
        WHERE    EXISTS ( SELECT   *
                          FROM     DEPENDENT AS D
                          WHERE    E.Ssn = D.Essn AND E.Sex = D.Sex
                                   AND E.Fname = D.Dependent_name);
```

Query 6. Retrieve the names of employees who have no dependents.

```
Q6:     SELECT   Fname, Lname
        FROM     EMPLOYEE
        WHERE    NOT EXISTS ( SELECT   *
                              FROM     DEPENDENT
                              WHERE    Ssn = Essn );
```

(a)

select fname, lname from employee except

(select fname, lname from employee, dependent where essn = ssn)

(b)

Select fname, lname from employee where ssn not in (select essn from dependent)

Eg

Query 7. List the names of managers who have at least one dependent.

```
Q7:   SELECT   Fname, Lname
      FROM     EMPLOYEE
      WHERE    EXISTS ( SELECT   *
                        FROM     DEPENDENT
                        WHERE    Ssn = Essn )
               AND
               EXISTS ( SELECT   *
                        FROM     DEPARTMENT
                        WHERE    Ssn = Mgr_ssn );
```

(a)

select distinct fname, lname from employee, department, dependent where ssn = mgr-ssn and ssn = essn;

1.4.1 For all / division is SQL

- Using EXISTS and NOT EXISTS

Q3: Retrieve the name of each employee who works on all the projects controlled by department number 5

Q3A:

```
SELECT   Fname, Lname
FROM     EMPLOYEE
WHERE    NOT EXISTS ( ( SELECT   Pnumber
                        FROM     PROJECT
                        WHERE    Dnum = 5)
                      EXCEPT   ( SELECT   Pno
                                 FROM     WORKS_ON
                                 WHERE    Ssn = Essn) );
```

leftover projects in dept 5 for emp

should not have leftover

all projects emp works in

Q3B:

```
SELECT   Lname, Fname
FROM     EMPLOYEE
WHERE    NOT EXISTS  ( SELECT   *
                       FROM     WORKS_ON B
                       WHERE    ( B.Pno IN  ( SELECT   Pnumber
                                              FROM     PROJECT
                                              WHERE    Dnum = 5 )
                       AND
                       NOT EXISTS  ( SELECT   *
                                     FROM     WORKS_ON C
                                     WHERE    C.Essn = Ssn
                                     AND      C.Pno = B.Pno )));
```

1.5 Explicit Set of values in where

Query 17. Retrieve the Social Security numbers of all employees who work on project numbers 1, 2, or 3.

```
Q17:    SELECT    DISTINCT Essn
        FROM      WORKS_ON
        WHERE     Pno IN (1, 2, 3);
```

1.6 JOIN in FROM

```
Q1A:    SELECT    Fname, Lname, Address
        FROM      (EMPLOYEE JOIN DEPARTMENT ON Dno = Dnumber)
        WHERE     Dname = 'Research';
```

- here also NULL ignored

1.6.1 Natural Join

```
Q1B:    SELECT    Fname, Lname, Address
        FROM      (EMPLOYEE NATURAL JOIN
                  (DEPARTMENT AS DEPT (Dname, Dno, Mssn, Msdate)))
        WHERE     Dname = 'Research';
```

- gives cartesian prod if col names do not match

1.6.2 Outer Joins

shows emps w/o super also ↙

```
Q8B:    SELECT    E.Lname AS Employee_name,
                  S.Lname AS Supervisor_name
        FROM      (EMPLOYEE AS E LEFT OUTER JOIN EMPLOYEE AS S
                  ON E.Super_ssn = S.Ssn);
```

Alternate syntax (Oracle only)

```
Q8C:   SELECT   E.Lname, S.Lname
       FROM     EMPLOYEE E, EMPLOYEE S
       WHERE    E.Super_ssn + = S.Ssn;
```

1.7 Aggregate Functions

Query 19. Find the sum of the salaries of all employees, the maximum salary, the minimum salary, and the average salary.

```
Q19:   SELECT   SUM (Salary), MAX (Salary), MIN (Salary), AVG (Salary)
       FROM     EMPLOYEE;
```

- NULL ignored from aggregate func on particular col
- count(*) includes NULL in few columns
- max, min, sum, avg, count

Query 20. Find the sum of the salaries of all employees of the 'Research' department, as well as the maximum salary, the minimum salary, and the average salary in this department.

```
Q20:   SELECT   SUM (Salary), MAX (Salary), MIN (Salary), AVG (Salary)
       FROM     (EMPLOYEE JOIN DEPARTMENT ON Dno = Dnumber)
       WHERE    Dname = 'Research';
```

- instead of where use having if group by used (filter out groups)

Queries 21 and 22. Retrieve the total number of employees in the company (Q21) and the number of employees in the 'Research' department (Q22).

```
Q21:   SELECT   COUNT (*)
       FROM     EMPLOYEE;

Q22:   SELECT   COUNT (*)
       FROM     EMPLOYEE, DEPARTMENT
       WHERE    DNO = DNUMBER AND DNAME = 'Research';
```

- select clause in group by is very imp (project)
- valid if 1 value per group

Query 24. For each department, retrieve the department number, the number of employees in the department, and their average salary.

```
Q24:    SELECT      Dno, COUNT (*), AVG (Salary)
        FROM        EMPLOYEE
        GROUP BY    Dno;
```

(a)

Fname	Minit	Lname	Ssn	...	Salary	Super_ssn	Dno
John	B	Smith	123456789		30000	333445555	5
Franklin	T	Wong	333445555		40000	888665555	5
Ramesh	K	Narayan	666884444		38000	333445555	5
Joyce	A	English	453453453	...	25000	333445555	5
Alicia	J	Zelaya	999887777		25000	987654321	4
Jennifer	S	Wallace	987654321		43000	888665555	4
Ahmad	V	Jabbar	987987987		25000	987654321	4
James	E	Bong	888665555		55000	NULL	1

Grouping EMPLOYEE tuples by the value of Dno

Dno	Count (*)	Avg (Salary)
5	4	33250
4	3	31000
1	1	55000

Result of Q24

- Group by —— NULL is one group
- Group by with having

Query 26. For each project *on which more than two employees work,* retrieve the project number, the project name, and the number of employees who work on the project.

```
Q26:    SELECT      Pnumber, Pname, COUNT (*)
        FROM        PROJECT, WORKS_ON
        WHERE       Pnumber = Pno
        GROUP BY    Pnumber, Pname
        HAVING      COUNT (*) > 2;
```

· Result

(b)

Pname	Pnumber	...	Essn	Pno	Hours
ProductX	1		123456789	1	32.5
ProductX	1		453453453	1	20.0
ProductY	2		123456789	2	7.5
ProductY	2		453453453	2	20.0
ProductY	2		333445555	2	10.0
ProductZ	3		666884444	3	40.0
ProductZ	3		333445555	3	10.0
Computerization	10	...	333445555	10	10.0
Computerization	10		999887777	10	10.0
Computerization	10		987987987	10	35.0
Reorganization	20		333445555	20	10.0
Reorganization	20		987654321	20	15.0
Reorganization	20		888665555	20	NULL
Newbenefits	30		987987987	30	5.0
Newbenefits	30		987654321	30	20.0
Newbenefits	30		999887777	30	30.0

These groups are not selected by the HAVING condition of Q26.

After applying the WHERE clause but before applying HAVING

Pname	Pnumber	...	Essn	Pno	Hours
ProductY	2		123456789	2	7.5
ProductY	2		453453453	2	20.0
ProductY	2		333445555	2	10.0
Computerization	10		333445555	10	10.0
Computerization	10	...	999887777	10	10.0
Computerization	10		987987987	10	35.0
Reorganization	20		333445555	20	10.0
Reorganization	20		987654321	20	15.0
Reorganization	20		888665555	20	NULL
Newbenefits	30		987987987	30	5.0
Newbenefits	30		987654321	30	20.0
Newbenefits	30		999887777	30	30.0

Pname	Count (*)
ProductY	3
Computerization	3
Reorganization	3
Newbenefits	3

Result of Q26
(Pnumber not shown)

After applying the HAVING clause condition

- Group by multiple

Query 27. For each project, retrieve the project number, the project name, and the number of employees from department 5 who work on the project.

```
Q27:    SELECT      Pnumber, Pname, COUNT (*)
        FROM        PROJECT, WORKS_ON, EMPLOYEE
        WHERE       Pnumber = Pno AND Ssn = Essn AND Dno = 5
        GROUP BY    Pnumber, Pname;
```

Q: Count the total number of employees whose salaries exceed $40,000 in each department, but only for departments where more than five employees work

incorrect:

execution order ↓

```
SELECT      Dno, COUNT (*)
FROM        EMPLOYEE
WHERE       Salary>40000
GROUP BY    Dno
HAVING      COUNT (*) > 5;
```

will first filter emp before counting dept size

correct:

```
Q28:    SELECT      Dno, COUNT (*)
        FROM        EMPLOYEE
        WHERE       Salary>40000 AND Dno IN
                    ( SELECT      Dno
                      FROM        EMPLOYEE
        GROUP BY    Dno
                      HAVING      COUNT (*) > 5)
        GROUP BY    Dno;
```

get dnos →

1.8 With and Case Clauses

- **WITH:** common table expressions (CTE)
  - write auxiliary statements
  - define temp table for one query

eg:
```
with abc as ← optional keyword
(
    complex query
        ⋮
)
select * from abc;
```

- WITH mainly for readibility ; can use nested query instead

```
Q28′:   WITH      BIGDEPTS (Dno) AS
                  ( SELECT      Dno
                    FROM        EMPLOYEE
                    GROUP BY    Dno
                    HAVING      COUNT (*) > 5)
        SELECT    Dno, COUNT (*)
        FROM      EMPLOYEE
        WHERE     Salary>40000 AND Dno IN BIGDEPTS
        GROUP BY  Dno;
```

- **CASE:** like switch case

```
U6′:    UPDATE    EMPLOYEE
        SET       Salary =
        CASE      WHEN    Dno = 5     THEN Salary + 2000
                  WHEN    Dno = 4     THEN Salary + 1500
                  WHEN    Dno = 1     THEN Salary + 3000
                  ELSE    Salary + 0 ;
```

Assertions & Triggers

2.1 ASSERTIONS

- Assertions: constraints outside scope of built-in relational model constraints

```
CREATE ASSERTION SALARY_CONSTRAINT
CHECK ( NOT EXISTS ( SELECT     *
                     FROM       EMPLOYEE E, EMPLOYEE M,
                                DEPARTMENT D
                     WHERE      E.Salary>M.Salary
                                AND     E.Dno = D.Dnumber
                                AND     D.Mgr_ssn = M.Ssn ) );
```

- Not in pg, only Oracle, MySQL

2.2 TRIGGERS

- Triggers: actions taken when event occurs
- Components: event(s), condition, action (ECA rule)
- Before or after event

```
R5: CREATE TRIGGER SALARY_VIOLATION
    BEFORE INSERT OR UPDATE OF SALARY, SUPERVISOR_SSN
      ON EMPLOYEE
    FOR EACH ROW
      WHEN ( NEW.SALARY > ( SELECT SALARY FROM EMPLOYEE
                            WHERE SSN = NEW.SUPERVISOR_SSN ) )
                            INFORM_SUPERVISOR(NEW.Supervisor_ssn,
                            NEW.Ssn );
```

action (function/procedure)
(can return) (no ret)

Functions

- \df → list
- Function: 0 or more params
- Procedural language for psql : plpgsql (we use)
  - can use others
- CREATE [OR REPLACE] FUNCTION name(params)
  RETURNS type
  LANGUAGE plpgsql
  AS

  $$
  declare
  -- variable declaration (local)
  begin
  -- logic
  end;
  $$

```
vibhamasti=# create table accounts (
vibhamasti(# id int generated by default as identity,
vibhamasti(# name varchar(100) not null,
vibhamasti(# balance dec(15, 2) not null,
vibhamasti(# primary key(id)
vibhamasti(# );
CREATE TABLE
vibhamasti=# insert into accounts(name, balance) values ('Bob', 10000);
INSERT 0 1
vibhamasti=# insert into accounts(name, balance) values ('Alice', 10000);
INSERT 0 1
vibhamasti=# insert into accounts(name, balance) values ('Colin', 10000);
INSERT 0 1
```

```
vibhamasti=# create function get_total(from_id int, to_id int)
returns int
language plpgsql
as
$$
declare
balance_total integer;
begin
select sum(balance) into balance_total from accounts where id between from_id an
d to_id; return balance_total;
end;
$$;
CREATE FUNCTION
```

```
vibhamasti=# select * from get_total(1, 2);
 get_total
-----------
     20000
(1 row)
```

Trigger

CREATE TRIGGER _____

BEFORE/AFTER _____

ON _____

FOR EACH ROW EXECUTE PROCEDURE

CREATE PROCEDURE

no return

cursor.call()

Call procedure cur.execute("CALL procedure(%s,%s)", ( )

cur.callproc ('func_name', (val1, val2))

VIEW

- Virtual table

```
V1:   CREATE VIEW    WORKS_ON1
      AS SELECT      Fname, Lname, Pname, Hours
         FROM        EMPLOYEE, PROJECT, WORKS_ON
         WHERE       Ssn = Essn AND Pno = Pnumber;

V2:   CREATE VIEW    DEPT_INFO(Dept_name, No_of_emps, Total_sal)
      AS SELECT      Dname, COUNT (*), SUM (Salary)
         FROM        DEPARTMENT, EMPLOYEE
         WHERE       Dnumber = Dno
         GROUP BY    Dname;
```

- Can make a view of a join

Strategies

(a) Query modification approach

- Maps view name to query (sub-query)

```
QV1:  SELECT    Fname, Lname
      FROM      WORKS_ON1
      WHERE     Pname = 'ProductX';
```


```
SELECT    Fname, Lname
FROM      EMPLOYEE, PROJECT, WORKS_ON
WHERE     Ssn = Essn AND Pno = Pnumber
          AND Pname = 'ProductX';
```

(b) View Materialisation / Realisation

- Stores view physically
- Updation strategies

- immediate update
- lazy update (most)
- periodic update

Updation of Views

■ A view with a single defining table is updatable if the view attributes contain the primary key of the base relation, as well as all attributes with the NOT NULL constraint that do not have default values specified.

■ Views defined on multiple tables using joins are generally not updatable.

■ Views defined using grouping and aggregate functions are not updatable.

UPDATE < > SET < > WHERE < >

View Constraints

CREATE VIEW < > AS < > WITH CHECK OPTION;

- Fails row constraint otherwise

View as an Authorisation Mechanism

- Restrict access to hidden details
- Grant permissions to users

Table 7.2 Summary of SQL Syntax

CREATE TABLE <table name> (<column name> <column type> [<attribute constraint>] { , <column name> <column type> [<attribute constraint>] } [<table constraint> { , <table constraint> }])
DROP TABLE <table name> ALTER TABLE <table name> ADD <column name> <column type>
SELECT [DISTINCT] <attribute list> FROM (<table name> { <alias> } | <joined table>) { , (<table name> { <alias> } | <joined table>) } [WHERE <condition>] [GROUP BY <grouping attributes> [HAVING <group selection condition>]] [ORDER BY <column name> [<order>] { , <column name> [<order>] }]
<attribute list> ::= (* | (<column name> | <function> (([DISTINCT] <column name> | *))) { , (<column name> | <function> (([DISTINCT] <column name> | *)) }))
<grouping attributes> ::= <column name> { , <column name> }
<order> ::= (ASC | DESC)
INSERT INTO <table name> [(<column name> { , <column name> })] (VALUES (<constant value> , { <constant value> }) { , (<constant value> { , <constant value> }) } | <select statement>)
DELETE FROM <table name> [WHERE <selection condition>]
UPDATE <table name> SET <column name> = <value expression> { , <column name> = <value expression> } [WHERE <selection condition>]
CREATE [UNIQUE] INDEX <index name> ON <table name> (<column name> [<order>] { , <column name> [<order>] }) [CLUSTER]
DROP INDEX <index name>
CREATE VIEW <view name> [(<column name> { , <column name> })] AS <select statement>
DROP VIEW <view name>

NOTE: The commands for creating and dropping indexes are not part of standard SQL.

OLAP

- Online Analytical Processing
- Allows data to be summarised and viewed in different ways in an online fashion
- Negligibe delay
- For explanation: schema
  `sales (item_name, color, clothes_size, quantity)`
- **Measure attributes:** measure some value that can be aggregated upon
  - eg: attribute quantity of the sales relation
- **Dimension attributes:** define the dimensions on which measure attributes are viewed
  - eg: attributes item_name, color and size of the sales relation
- Data that can be modelled as dimension attributes and measure attributes are called **multidimensional data**

Suppose that *item_name* can take on the values (skirt, dress, shirt, pants), *color* can take on the values (dark, pastel, white), *clothes_size* can take on values (small, medium, large), and *quantity* is an integer value representing the total number of items of a given {*item_name*, *color*, *clothes_size* }. An instance of the *sales* relation is shown in Figure 5.16.

item_name	color	clothes_size	quantity
skirt	dark	small	2
skirt	dark	medium	5
skirt	dark	large	1
skirt	pastel	small	11
skirt	pastel	medium	9
skirt	pastel	large	15
skirt	white	small	2
skirt	white	medium	5
skirt	white	large	3
dress	dark	small	2
dress	dark	medium	6
dress	dark	large	12
dress	pastel	small	4
dress	pastel	medium	3
dress	pastel	large	3
dress	white	small	2
dress	white	medium	3
dress	white	large	0
shirt	dark	small	2
shirt	dark	medium	6
shirt	dark	large	6
shirt	pastel	small	4
shirt	pastel	medium	1
shirt	pastel	large	2
shirt	white	small	17
shirt	white	medium	1
shirt	white	large	10
pants	dark	small	14
pants	dark	medium	6
pants	dark	large	0
pants	pastel	small	1
pants	pastel	medium	0
pants	pastel	large	1
pants	white	small	3
pants	white	medium	0
pants	white	large	2

Figure 5.16 An example of *sales* relation.

item_name	color	clothes_size	quantity
skirt	dark	**all**	8
skirt	pastel	**all**	35
skirt	white	**all**	10
skirt	**all**	**all**	53
dress	dark	**all**	20
dress	pastel	**all**	10
dress	white	**all**	5
dress	**all**	**all**	35
shirt	dark	**all**	14
shirt	pastel	**all**	7
shirt	white	**all**	28
shirt	**all**	**all**	49
pants	dark	**all**	20
pants	pastel	**all**	2
pants	white	**all**	5
pants	**all**	**all**	27
all	dark	**all**	62
all	pastel	**all**	54
all	white	**all**	48
all	**all**	**all**	164

Figure 5.21 Relational representation of the data in Figure 5.17.

1. CROSS-TABULATION / PIVOT TABLE

- Row headers — values of one attr
  Column headers — values of another attr

clothes_size **all**

		color			
		dark	pastel	white	total
item_name	skirt	8	35	10	53
	dress	20	10	5	35
	shirt	14	7	28	49
	pants	20	2	5	27
	total	62	54	48	164

Figure 5.17 Cross tabulation of *sales* by *item_name* and *color*.

2. DATA CUBE

- All combinations with summarisations
- Here: measure attribute = quantity

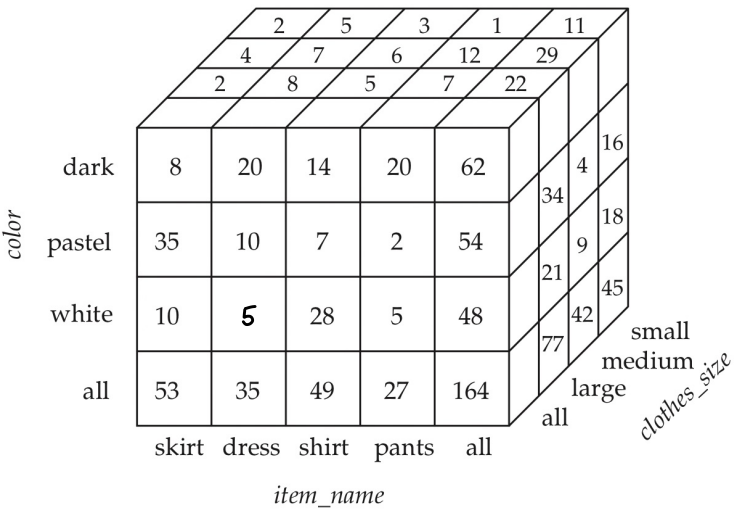

Figure 5.18 Three-dimensional data cube.

- Number of ways to group tuples for aggregation
  $3 \times 3 \times 4 = 36$
- With all
  $4 \times 4 \times 5 = 80$
- For n dimensions, all value combinations on every group in the power set 2^n are computed for

  $2^3 = 8$ groups
  group by on each group

3. PIVOT

- Cross-tab is 2D view
- Change dimensions used in cross-tab

4. SLICING

- Slice of data cube

5. ROLL UP

- Fine granularity → coarse granularity

6. DRILL DOWN

- Coarse granularity → fine granularity
- Must be generated

Implementation of OLAP

- MOLAP : Multidimensional attr
- ROLAP : Relational
- HOLAP : Hybrid

```
select item_name, color, clothes_size, sum(quantity)
from sales
group by rollup(item_name, color, clothes_size);
```