卐 DBMS Short Revision Notes 卐

Sourabh Aggarwal

Compiled on January 26, 2019

Contents

1 Doubts

discriminator of weak entity? (or did mam say that she is not going into details of it?)
Slide 15 of lect4.
Does count avoid null (to be asked from myself, q: 21 from lab 3.) in specialisation shouldnt the top level has certain fixed attributes? (I mean when we merge specialised (low level) to above it gives us certain attributes which doesn't match when merging others). Though it oculd be very well the case that we are getting different tables after merging.

2 Intro

Database is simply a collection of related info.
Create Read (Retrieve) Update Delete (CRUD)
Two types of Databases:-

- Relational Databases (SQL): Organize data into one or more tables, each table has columns and rows and a unique key identifies each row.
- Non Relational (noSQL / not just SQL): Organize data is anything but a traditional table. Like key value stores, Documents (JSON), Graphs, etc.

Relational database management systems (RDBMS): softwares which help users create and maintain a a relational database. Ex: mySQL
Schema: Is just an overall structure of our database, columns, their types etc.
SQL (Standard Query Language): Standardized language for interacting with RDBMS. It's basically 4 types of languages in one:

- Data Query Language (DQL)
- Data Definition Language (schemas etc.)
- Data Control Language (permissions etc.)
- Data Manipulation Language (update etc.)

Queries are requests made to the database management systems for specific information.
Primary Key: Column which uniquely identifies each row. (It cannot contain NULL values.) Tables are limited to ONE primary key each.
Surrogate key: Primary key which has no inference, just a random no.
Natural Key: Aadhaar no., etc. which have real world inference/mapping. (it is a primary key)
Composite Key: Key which requires 2 or more attributes, Primary key can be a composite key.
Foreign Key: Attribute which will link us to another database table. Foreign key stores primary key of a row of another database table. A particular table can have more than one foreign key. And it can have NULL values. Primary key could be a composite of foreign keys.

2.1 SQL

```
create database girrafe;

INT -- whole numbers
DECIMAL(M, N) -- M is the total no. of digits and N is the no.
of digits after the decimal point.
CHAR(l) -- String of fixed length l.
VARCHAR(l) -- String of maximum length l.
BLOB -- Binary Large Object
DATE -- 'YYYY-MM-DD'
TIMESTAMP -- 'YYYY-MM-DD HH:MM:SS'


CREATE TABLE student (
    student_id INT PRIMARY KEY,  -- now automatically student_id
    can't be null.
    name VARCHAR (30),
    major VARCHAR (20)
);

DESCRIBE student;  -- Describes our table

DROP TABLE student;  -- Deletes our table

ALTER TABLE student ADD gpa DECIMAL (3, 2) default 0;  -- Add a
column to our table

alter table student add gender enum ('M', 'F') not null;  --
will give some default value (like 'M') to already existing rows
```

```sql
-- aliter
alter table student add gender varchar (1) check (gender in
('M', 'F')) not null

ALTER TABLE student DROP COLUMN gpa;  -- drops our gpa column

SELECT * FROM student;  -- show all rows

SELECT '123'; -- will create at table with only one column
'123', and only one entry '123'.

INSERT INTO student VALUES (1, 'Jack', 'Biology');  -- add this
row, parameters should be given in order

INSERT INTO student (student_id, name) VALUES (2, 'Kate');  --
Now we need not include 'major', it will show 'NULL' in major
now.
```

```sql
UPDATE student
SET major = 'Bio'
WHERE major = 'Biology';  -- other operators are <> (not equal),
> (greater), <, >=, <=. our target is to update the major name
to Bio in case the major name is Biology
-- Or we could have done WHERE student_id > 3;
-- Or SET major = 'Biochemistry'
-- WHERE major = 'Bio' OR major = 'Chemistry';
-- SET name = 'Tom', major = 'undecided'
-- WHERE student_id = 1;
-- Note: If we remove WHERE then it will affect all of the rows.
DELETE FROM student -- if we put semicolon at the end of this
statement then it will delete all of the rows in the table
WHERE name = 'Tom' AND major = 'undecided';
------------------
SELECT name, major
FROM student
ORDER by name DESC; -- will give the entries in the descending
order of names. If we remove DESC then it will be order by
ascending order. btw ASC can as well be used instead.
-- ORDER by major, student_id DESC; -- will order by major and
in case there is a tie then they will be ordered by descending
student_id.
-- We can add LIMIT 2; this would give us only 2 entries.
-- We can do LIMIT 2 OFFSET 5; which will give entry 6, 7.
-- select * from takes order by field (grade, 'S', 'A+', 'A',
'A-', ..., 'D-'); to order with specified order.
-- aliter is given below, suppose the order is 2, 1, 3.
select * from people,
where id in (1, 2, 3)
order by case id
when 2 then 0
when 1 then 1
when 3 then 2
else 3 then END;

SELECT name, major
FROM student
WHERE name IN ('Claire', 'Kate', 'Mike') AND student_id > 2; --
IN checks for set membership
```

Company Database

Employee

emp_id	first_name	last_name	birth_date	sex	salary	super_id	branch_id
100	David	Wallace	1967-11-17	M	250,000	NULL	1
101	Jan	Levinson	1961-05-11	F	110,000	100	1
102	Michael	Scott	1964-03-15	M	75,000	100	2
103	Angela	Martin	1971-06-25	F	63,000	102	2
104	Kelly	Kapoor	1980-02-05	F	55,000	102	2
105	Stanley	Hudson	1958-02-19	M	69,000	102	2
106	Josh	Porter	1969-09-05	M	78,000	100	3
107	Andy	Bernard	1973-07-22	M	65,000	106	3
108	Jim	Halpert	1978-10-01	M	71,000	106	3

Branch

branch_id	branch_name	mgr_id	mgr_start_date
1	Corporate	100	2006-02-09
2	Scranton	102	1992-04-06
3	Stamford	106	1998-02-13

Client

client_id	client_name	branch_id
400	Dunmore Highschool	2
401	Lackawana Country	2
402	FedEx	3
403	John Daly Law, LLC	3
404	Scranton Whitepages	2
405	Times Newspaper	3
406	FedEx	2

Works_With

emp_id	client_id	total_sales
105	400	55,000
102	401	267,000
108	402	22,500
107	403	5,000
108	403	12,000
105	404	33,000
107	405	26,000
102	406	15,000
105	406	130,000

Branch Supplier

branch_id	supplier_name	supply_type
2	Hammer Mill	Paper
2	Uni-ball	Writing Utensils
3	Patriot Paper	Paper
2	J.T. Forms & Labels	Custom Forms
3	Uni-ball	Writing Utensils
3	Hammer Mill	Paper
3	Stamford Lables	Custom Forms

Labels

(red)	Primary Key
(green)	Foreign Key
(blue)	Attribute

```sql
CREATE TABLE employee (
  emp_id INT PRIMARY KEY,
  first_name VARCHAR(40),
  last_name VARCHAR(40),
  birth_day DATE,
  sex VARCHAR(1),
  salary INT,
  super_id INT,
  branch_id INT
);

CREATE TABLE branch (
  branch_id INT PRIMARY KEY,
  branch_name VARCHAR(40),
  mgr_id INT,
  mgr_start_date DATE,
  FOREIGN KEY(mgr_id) REFERENCES employee(emp_id) ON DELETE SET
  NULL
  -- example of multiple foreign key:
  -- foreign key (course_id, sec_id, semester) references
  section (course_id, sec_id, semester) on delete cascade
  -- foreign key (ID) references student (ID) on delete cascade
);

ALTER TABLE employee
ADD FOREIGN KEY(branch_id)
REFERENCES branch(branch_id)
ON DELETE SET NULL;

ALTER TABLE employee
ADD FOREIGN KEY(super_id)
REFERENCES employee(emp_id)
ON DELETE SET NULL;

CREATE TABLE client (
  client_id INT PRIMARY KEY,
  client_name VARCHAR(40),
  branch_id INT,
  FOREIGN KEY(branch_id) REFERENCES branch(branch_id) ON DELETE
  SET NULL
);

CREATE TABLE works_with (
  emp_id INT,
```

```sql
  client_id INT,
  total_sales INT,
  PRIMARY KEY(emp_id, client_id),  -- in this way we can set
  multiple primary keys.
  FOREIGN KEY(emp_id) REFERENCES employee(emp_id) ON DELETE
  CASCADE,
  FOREIGN KEY(client_id) REFERENCES client(client_id) ON DELETE
  CASCADE
);

CREATE TABLE branch_supplier (
  branch_id INT,
  supplier_name VARCHAR(40),
  supply_type VARCHAR(40),
  PRIMARY KEY(branch_id, supplier_name),
  FOREIGN KEY(branch_id) REFERENCES branch(branch_id) ON DELETE
  CASCADE
);


--
-----------------------------------------------------------------

-- Corporate
INSERT INTO employee VALUES(100, 'David', 'Wallace',
'1967-11-17', 'M', 250000, NULL, NULL);

INSERT INTO branch VALUES(1, 'Corporate', 100, '2006-02-09');

UPDATE employee
SET branch_id = 1
WHERE emp_id = 100;

INSERT INTO employee VALUES(101, 'Jan', 'Levinson',
'1961-05-11', 'F', 110000, 100, 1);

-- Scranton
INSERT INTO employee VALUES(102, 'Michael', 'Scott',
'1964-03-15', 'M', 75000, 100, NULL);

INSERT INTO branch VALUES(2, 'Scranton', 102, '1992-04-06');

UPDATE employee
SET branch_id = 2
WHERE emp_id = 102;

INSERT INTO employee VALUES(103, 'Angela', 'Martin',
'1971-06-25', 'F', 63000, 102, 2);
INSERT INTO employee VALUES(104, 'Kelly', 'Kapoor',
'1980-02-05', 'F', 55000, 102, 2);
INSERT INTO employee VALUES(105, 'Stanley', 'Hudson',
'1958-02-19', 'M', 69000, 102, 2);

-- Stamford
INSERT INTO employee VALUES(106, 'Josh', 'Porter',
'1969-09-05', 'M', 78000, 100, NULL);

INSERT INTO branch VALUES(3, 'Stamford', 106, '1998-02-13');

UPDATE employee
SET branch_id = 3
WHERE emp_id = 106;

INSERT INTO employee VALUES(107, 'Andy', 'Bernard',
'1973-07-22', 'M', 65000, 106, 3);
INSERT INTO employee VALUES(108, 'Jim', 'Halpert',
'1978-10-01', 'M', 71000, 106, 3);


-- BRANCH SUPPLIER
INSERT INTO branch_supplier VALUES(2, 'Hammer Mill', 'Paper');
INSERT INTO branch_supplier VALUES(2, 'Uni-ball', 'Writing
Utensils');
INSERT INTO branch_supplier VALUES(3, 'Patriot Paper',
'Paper');
INSERT INTO branch_supplier VALUES(2, 'J.T. Forms & Labels',
'Custom Forms');
INSERT INTO branch_supplier VALUES(3, 'Uni-ball', 'Writing
Utensils');
INSERT INTO branch_supplier VALUES(3, 'Hammer Mill', 'Paper');
INSERT INTO branch_supplier VALUES(3, 'Stamford Lables',
'Custom Forms');
```

```sql
-- CLIENT
INSERT INTO client VALUES(400, 'Dunmore Highschool', 2);
INSERT INTO client VALUES(401, 'Lackawana Country', 2);
INSERT INTO client VALUES(402, 'FedEx', 3);
INSERT INTO client VALUES(403, 'John Daly Law, LLC', 3);
INSERT INTO client VALUES(404, 'Scranton Whitepages', 2);
INSERT INTO client VALUES(405, 'Times Newspaper', 3);
INSERT INTO client VALUES(406, 'FedEx', 2);

-- WORKS_WITH
INSERT INTO works_with VALUES(105, 400, 55000);
INSERT INTO works_with VALUES(102, 401, 267000);
INSERT INTO works_with VALUES(108, 402, 22500);
INSERT INTO works_with VALUES(107, 403, 5000);
INSERT INTO works_with VALUES(108, 403, 12000);
INSERT INTO works_with VALUES(105, 404, 33000);
INSERT INTO works_with VALUES(107, 405, 26000);
INSERT INTO works_with VALUES(102, 406, 15000);
INSERT INTO works_with VALUES(105, 406, 130000);

-- Find out all the different genders
SELECT DISCINCT sex
FROM employee;

-- Find all employee's id's and names who were born after 1969
SELECT emp_id, first_name, last_name
FROM employee
WHERE birth_day >= 1970-01-01;

-- Find all employees born between 1970 and 1975
SELECT *
FROM employee
WHERE birth_day BETWEEN '1970-01-01' AND '1975-01-01';

-- Functions

-- Find the number of employees
SELECT COUNT(super_id)
FROM employee;

-- Find the average of all employee's salaries
SELECT AVG(salary)
FROM employee;

-- Find the sum of all employee's salaries
SELECT SUM(salary)
FROM employee;

-- Aggregation.
-- Find out how many males and females there are
SELECT COUNT(sex), sex  -- now we have two columns, viz,
COUNT(sex) and sex.
FROM employee
GROUP BY sex

-- Find the total sales of each salesman
SELECT SUM(total_sales), emp_id
FROM works_with
GROUP BY emp_id;
-- IMP NOTE: sum does not avoid null values, null + something is
always null so better check for null values.
select sum(tot_cred) from student where tot_cred is not null;

-- Find the total amount of money spent by each client
SELECT SUM(total_sales), client_id
FROM works_with
GROUP BY client_id;

select dept_name from department, (select max(budget) as MX
from department) as T where budget = T.MX;

select dept_name, max(AG) from (select dept_name, avg(salary)
as AG from instructor group by dept_name) as M;

select name, salary, dept_name from instructor where salary >
all (select salary from instructor where dept_name in
('Biology', 'History', 'Finance'));

-- Wildcards

-- % = any # characters, _ = one character
```

```sql
-- Find any employee born on the 10th day of the month
SELECT *
FROM employee
WHERE birth_day LIKE '_____10%';

-- Find any clients who are schools
SELECT *
FROM client
WHERE client_name LIKE '%Highschool%';

-- Union

-- Find a list of all clients & branch suppliers' names
SELECT client.client_name AS Non-Employee_Entities,
client.branch_id AS Branch_ID
FROM client
UNION -- both above and below thing should have same number of
columns and same data type respectively
SELECT branch_supplier.supplier_name, branch_supplier.branch_id
FROM branch_supplier;

-- JOINS (used to combine rows from two or more tables based on
the related columns)

-- Add the extra branch
INSERT INTO branch VALUES(4, "Buffalo", NULL, NULL);  -- here I
have used double quotes but strings can as well be represented
with single quotes.

-- Find all branches and the names of their managers
SELECT employee.emp_id, employee.first_name, branch.branch_name
FROM employee
JOIN branch    -- LEFT JOIN (when we use LEFT JOIN all the rows
from the left table gets included as well, similarly for RIGHT
JOIN), RIGHT JOIN  (we added buffalo just so that we can see
difference in case of RIGHT JOIN)
ON employee.emp_id = branch.mgr_id;
-- we can add a where clause beneath this like where
branch.mgr_start_date > something

-- Show ID, name, courses, and corresponding credits of each
student.
select M.ID, M.name, C.title, C.credits from (select S.ID,
S.name, T.course_id from student as S join takes as T on S.ID =
T.ID) as M join course as C on M.course_id = C.course_id;

-- nested queries

-- Find names of all employees who have sold over 50,000
SELECT employee.first_name, employee.last_name
FROM employee
WHERE employee.emp_id IN (SELECT works_with.emp_id  -- note
that first sql will execute the part which is inside "()"
                          FROM works_with
                          WHERE works_with.total_sales >
                          50000);

-- Find all clients who are handled by the branch that Michael
Scott manages
-- Assume you know Michael's ID
SELECT client.client_id, client.client_name
FROM client
WHERE client.branch_id = (SELECT branch.branch_id
                          FROM branch
                          WHERE branch.mgr_id = 102);

 -- Find all clients who are handles by the branch that Michael
 Scott manages
 -- Assume you DONT'T know Michael's ID
 SELECT client.client_id, client.client_name
 FROM client
 WHERE client.branch_id = (SELECT branch.branch_id
                           FROM branch
                           WHERE branch.mgr_id = (SELECT
                           employee.emp_id
                                                 FROM employee
                                                 WHERE
                                                 employee.first_name
                                                 = 'Michael'
                                                 AND
                                                 employee.last_name
                                                 ='Scott'
                                                 LIMIT 1));


-- Find the names of employees who work with clients handled by
the scranton branch
SELECT employee.first_name, employee.last_name
FROM employee
WHERE employee.emp_id IN (
                         SELECT works_with.emp_id
                         FROM works_with
                         )
AND employee.branch_id = 2;

-- Find the names of all clients who have spent more than
100,000 dollars
SELECT client.client_name
FROM client
WHERE client.client_id IN (
                          SELECT client_id
                          FROM (
                                SELECT
                                SUM(works_with.total_sales) AS
                                totals, client_id
                                FROM works_with
                                GROUP BY client_id) AS
                                total_client_sales
                          WHERE totals > 100000
);

-- Correct the tot_credit attributes for each tuple in student
table such that total credit is
equal to the credit of courses successfully completed by the
student. Here successfully
completed means student has a grade that is not 'F' . Students
who did not take any
courses, the output for them should show total credit 0.

update student as ST, (select B.ID, coalesce(B.crd, 0) as cred
from (select J.ID, sum(C.credits) as crd from (select N.ID,
N.grd, N.course_id from (select M.ID, coalesce(M.grade, 'N') as
grd, M.course_id from (select S.ID, S.name, T.course_id,
T.grade from student as S left join takes as T on S.ID = T.ID)
as M) as N) as J left join course as C on J.course_id =
C.course_id where J.grd <> 'F' group by J.ID) as B) as TEMP set
tot_cred = TEMP.cred where ST.ID = TEMP.ID;

-- deleting entries in the table when they have foreign keys
associated to them
-- ON DELETE NULL means that say if we delete Michael Scott then
second entry in the branch table will get mgr_id set to NULL
whereas in ON DELETE CASCADE the entire row will get deleted in
branch table. Use ON DELETE CASCADE when that foreign key is
very important like say it forms a primary key.

-- Views

-- Create view of students who are studying in a department with
budget more than 80000

create view stud_dept_budg as select S.ID, S.name, S.dept_name,
D.budget from student as S join department as D on S.dept_name
= D.dept_name where D.budget > 80000;

-- Now if we delete (or dec budget) some department with budget
> 80000. and then do select * from stud_dept_budg; it wont show
name corresponding to that department.
-- CREATE
--     TRIGGER `event_name` BEFORE/AFTER INSERT/UPDATE/DELETE
--     ON `database`.`table`
--     FOR EACH ROW BEGIN
--     -- trigger body
--     -- this code is applied to every
--     -- inserted/updated/deleted row
--     END;
```

```
CREATE TABLE trigger_test (
     message VARCHAR(100)
);


DELIMITER $$  -- changing delimiter from semicolon to ££ as we
will be using semicolon inside and we don't want sql to think
that when we put semicolon we are done with our trigger
CREATE
    TRIGGER my_trigger BEFORE INSERT
    ON employee
    FOR EACH ROW BEGIN  -- for each new item that is inserted
        INSERT INTO trigger_test VALUES('added new employee');
    END$$
DELIMITER ;  -- changing delimiter back to semi colon.
INSERT INTO employee
VALUES(109, 'Oscar', 'Martinez', '1968-02-19', 'M', 69000, 106,
3);


DELIMITER $$
CREATE
    TRIGGER my_trigger BEFORE INSERT
    ON employee
    FOR EACH ROW BEGIN
        INSERT INTO trigger_test VALUES(NEW.first_name);
    END$$
DELIMITER ;
INSERT INTO employee
VALUES(110, 'Kevin', 'Malone', '1978-02-19', 'M', 69000, 106,
3);

DELIMITER $$
CREATE
    TRIGGER my_trigger BEFORE INSERT
    ON employee
    FOR EACH ROW BEGIN
         IF NEW.sex = 'M' THEN
               INSERT INTO trigger_test VALUES('added male
               employee');
         ELSEIF NEW.sex = 'F' THEN
               INSERT INTO trigger_test VALUES('added female');
         ELSE
               INSERT INTO trigger_test VALUES('added other
               employee');
         END IF;
    END$$
DELIMITER ;
INSERT INTO employee
VALUES(111, 'Pam', 'Beesly', '1988-02-19', 'F', 69000, 106, 3);


DROP TRIGGER my_trigger;
```

Around last 20 mins of that video show how to convert ER Diagram to actual database schema.
Entity = An object we want to model and store information about
Attributes = Specific pieces of information about an entity
Composite attribute = An attribute that can be broken up into sub attributes
Multi - valued attribute = An attribute than can have more than one value (like same student can have more than one club but only one GPA)
Derived Attribute = An attribute that can be derived from the other attributes.
ER = Entity Relationship defines a relationship between two entities.
Relationship Attribute = An attribute about the relationship

2.2 Lect1

Levels of abstraction
Physical level: describes how a record (e.g., instructor) is stored.
Logical level: describes data stored in database, and the relationships among the data.
View level: application programs hide details of data types. Views can also hide information (such as an employee's salary) for security purposes.
Logical Schema: the overall logical structure of the database Example: The database consists of information about a set of customers and accounts in a bank and the relationship between them Analogous to type information of a variable in a program
Physical schema: the overall physical structure of the database
Instance: the actual content of the database at a particular point in time Analogous to the value of a variable
Physical Data Independence: the ability to modify the physical schema without changing the logical schema
Data Models: A collection of tools for describing Data, Data relationships, Data semantics, Data constraints. Ex: Relationship Model, Entity relationship model, object based model.
Database engine:
1) Storage manager: is a program module that provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system.
2) Query Processing
3) Transaction Management:-
Transaction: is a collection of operations that performs a single logical function in a database application
Transaction-management component: ensures that the database remains in a consistent (correct) state despite system failures (e.g., power failures and operating system crashes) and transaction failures.

2.3 Lect2

Lecture was about relational model
K is a superkey of R if values for K are sufficient to identify a unique tuple of each possible relation r of R.
Superkey K is a candidate key if K is minimal
One of the candidate key is selected to be the primary key

2.4 Lect3

An entity is a "thing" or "object" in the real world that is distinguishable from other objects.
Entities are described in a database by a set of attributes
A relationship is an association among several entities
The set of all entities of the same type and the set of all relationships of the same type are termed an entity set and relationship set, respectively.
A relationship may also have attributes called descriptive attributes
Domain: the set of permitted values for each attribute
An entity set that does not have sufficient attributes to form a primary key is termed a weak entity set
An entity set that has a primary key is termed a strong entity set
For a weak entity set to be meaningful, it must be associated with another entity set, called the identifying or owner entity set
Every weak entity must be associated with an identifying entity; that is, the weak entity set is said to be existence dependent on the identifying entity set. The identifying entity set is said to own the weak entity set that it identifies.
The relationship associating the weak entity set with the identifying entity set is called the identifying relationship
The discriminator (or partial key) of a weak entity set is the set of attributes that distinguishes among all the entities of a weak entity set.
The primary key of a weak entity set is formed by the primary key of the strong entity set on which the weak entity set is existence dependent, plus the weak entity set's discriminator

2.5 Lect4-5

See slides.

2.6 Lect6

See slides.