Review of Last Lecture

- Lecture 6-2: Midterm
- ORDER BY
- GROUP BY, HAVING
  - Special requirements
- UNION, INTERSECT, MINUS
  - Special requirements
- How to process NULL
  - IS NULL, IS NOT NULL
  - 3 value logic
  - Where clause treats UNKNOWN as false, but UNKNOWN is not equal to FALSE.
  - Aggregation with NULL
    - COUNT(*)

In-class Exercise

- Find the pair of employees who have worked together on any project (Bonus: a. get their names; b. remove duplicate like (e1, e2) and (e2, e1))

Today's Plan

- Data definition
  - Create, remove and change schema
- Data modification
  - Create, remove and change an instance

SQL DEFINITION AND MODIFICATION

CHAPTER 05

Michael Liu

University of Waterloo

Reading Assignment for This Chapter

- Textbook Chapter 08 and 09

PART II: DATA DEFINITION

Data Definition

- A database schema comprises declarations for the relations (“tables”) of the database.
- Many other kinds of elements may also appear in the database schema, including views, indexes, and triggers, which we’ll introduce later.
- Three basic operations
  - Create, Drop and Alter

Create Table

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

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

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

- Example:

  create table *branch*
  (*branch-name* char(15) **not null,**
  branch-city char(30),
  assets integer)

Domain Type (FYI)

- **char(n).** Fixed length character string, with user-specified length *n.*
- **varchar(n).** Variable length character strings, with user-specified maximum length *n.*
- **int.** Integer (a finite subset of the integers that is machine-dependent).
- **smallint.** Small integer (a machine-dependent subset of the integer domain type).
- **numeric(p,d).** Fixed point number, with user-specified precision of *p* digits, with d digits to the right of decimal point.
- **real, double precision.** Floating point and double-precision floating point numbers, with machine-dependent precision.
- **float(n).** Floating point number, with user-specified precision of at least *n* digits.
- Null values are allowed in all the domain types. Declaring an attribute to be **not null** prohibits null values for that attribute.
- **create domain** construct in SQL-92 creates user-defined domain types

  create domain *person-name* **char**(20) **not null**

Date/Time Type (FYI)

- **date.** Dates, containing a (4 digit) year, month and date
  - E.g. **date** '2001-7-27'
- **time.** Time of day, in hours, minutes and seconds.
  - E.g. **time** '09:00:30' **time** '09:00:30.75'
- **timestamp**: date plus time of day
  - E.g. **timestamp** '2001-7-27 09:00:30.75'
- **Interval**: period of time
  - E.g. Interval '1' day
  - Subtracting a date/time/timestamp value from another gives an interval value
  - Interval values can be added to date/time/timestamp values
- Can extract values of individual fields from date/time/timestamp
  - E.g. **extract** (**year from** r.starttime)
- Can cast string types to date/time/timestamp
  - E.g. **cast** <string-valued-expression> **as date**

Integrity Constraint in SQL I

- **not null**
- **primary key** (A_1, ..., A_n)
- **check** *(P)*, where *P* is a predicate
  - P must be satisfied by all tuples

Example: Declare *branch-name* as the primary key for *branch* and ensure that the values of *assets* are non-negative.

```
create table branch
    (branch-name char(15),
     branch-city  char(30)
     assets       integer,
     primary key (branch-name),
     check (assets >= 0))
```

primary key declaration on an attribute automatically ensures **not null** in SQL-92 onwards, needs to be explicitly stated in SQL-89

PRIMARY KEY vs. UNIQUE

- There can be only one PRIMARY KEY for a relation, but several UNIQUE attributes.
- No attribute of a PRIMARY KEY can ever be NULL in any tuple.
- But attributes declared UNIQUE may have NULL's, and there may be several tuples with NULL.

Other Declarations for Attributes

- Two other declarations we can make for an attribute are:
  - NOT NULL means that the value for this attribute may never be NULL.
  - DEFAULT <value> says that if there is no specific value known for this attribute's component in some tuple, use the stated <value>.

Example: Default Values

```
CREATE TABLE Drinkers (
        name  CHAR(30) PRIMARY KEY,
        addr   CHAR(50)
               DEFAULT '123 Sesame St.',
        phone CHAR(16)
);
```

```
INSERT INTO Drinkers(name)
VALUES('Sally');
```

Effect of Defaults

- Suppose we insert the fact that Sally is a drinker, but we know neither her address nor her phone.
- An INSERT with a partial list of attributes makes the insertion possible:

```
INSERT INTO Drinkers(name)
VALUES('Sally');
```

name	addr	Phone
'Sally'	'123 Sesame St'	NULL

Integrity Constraint in SQL II - FK

FOREIGN KEY (A1, A2 … …) REFERENCES R(B1, B2 .. …)

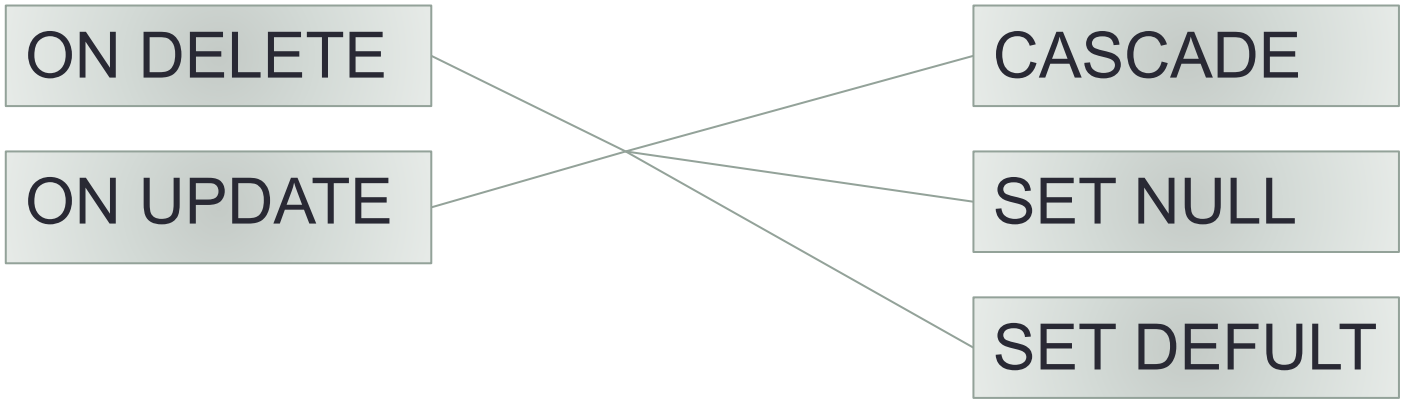
```
CREATE TABLE   EMP (
                ENAME          VARCHAR(30)  NOT NULL,
                ESSN           CHAR(9),
                BDATE          DATE,
                DNO            INTEGER  DEFAULT 1,
                SUPERSSN       CHAR(9),

                PRIMARY KEY (ESSN),

                FOREIGN KEY (DNO) REFERENCES DEPT(DNUMBER)
                ON DELETE SET DEFAULT ON UPDATE CASCADE,

                CONSTRAINT FK1
                FOREIGN KEY (SUPERSSN) REFERENCES EMP
                ON DELETE SET NULL ON UPDATE CASCADE);
```

Integrity Constraint in SQL III



FOREIGN KEY (SID) REFERENCE Students(SID)

Enrolled

sid	cid	grade
53666	Carnatic101	C
53666	Reggae203	B
53650	Topology112	A
53666	History105	B

Students

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

Alter and Drop Table I

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

  alter table *r* **add** *A D*

  where *A* is the name of the attribute to be added to relation *r* and *D* is the domain of *A*.
  - All tuples in the relation are assigned *null* as the value for the new attribute.

Alter and Drop Table II

- The **alter table** command can also be used to drop attributes of a relation

  alter table *r* **drop** *A*

  where *A* is the name of an attribute of relation *r*
  - Dropping of attributes not supported by many databases
- The **alter table** command can also be used to drop or add constraints

Examples of Drop and Alter

```
ALTER TABLE Bars ADD phone CHAR(16)
  DEFAULT ‘unlisted’;

DROP TABLE  Dependent;

ALTER TABLE Bars DROP license;
```

PART III DATA MODIFICATIONS

Database Modifications

- A modification command does not return a result (as a query does), but changes the database in some way.
  - Modify its content but not schema
- Three kinds of modifications:
  - Insert a tuple or tuples.
  - Delete a tuple or tuples.
  - Update the value(s) of an existing tuple or tuples.

Delete

- Delete whole tuples, one relation at a time

```
DELETE    FROM    R
WHERE     P;
```

- Find all tuples *t* in *R* such that *P(t)* is true, then remove *t* from *R*.

Delete: Example

- Delete all employees in Department #2

```
DELETE    FROM      Employee
WHERE     DNO = 2;
```

- Delete all employees working for the research department

```
DELETE    FROM      Employee
WHERE     DNO IN
          (SELECT   DNumber
          FROM      Department
          WHERE     DName = "Research");
```

Delete: Example

```
/* Delete student with the lowest grade */
DELETE FROM CS338Grades
WHERE grade in
        (SELECT MIN(grade)
        FROM CS338Grades)
```

Problem: as we delete tuples, the lowest grade changes

CS338Grades

SID	grade
001	70
002	80
003	90

Delete: 2 Step Process

- Delete (for insert and update as well) is done in two steps
  1. First, identify the tuples to be deleted using the criteria and mark them
  2. Second, delete all the marked tuples (without re-computing the criteria).

```
/* Delete student with the lowest grade */
DELETE FROM CS338Grades
WHERE grade in
        (SELECT MIN(grade)
        FROM CS338Grades)
```

CS338Grades

SID	grade
001	70
002	80
003	90

Delete: Exercise

- Delete from Works_ON entries where are more than one employee work on the project.

```
DELETE FROM Works_ON
WHERE PNO IN (
  SELECT PNO FROM Works_ON
  GROUP BY PNO
  HAVING     COUNT(ESSN) > 1);
```

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

Insert

- Insert a tuple or the results of a query
- Add a new tuple to *account*

  insert into *account*
  values ('A-9732', 'Perryridge',1200) ---- correct order!

or equivalently

  insert into

  account (branch-name, balance, account-number)
  values ('Perryridge', 1200, 'A-9732')

- Add a new tuple to *account* with *balance* set to null

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

What if we don't provide a value for a particular attribute during INSERT?

- Use default value if there is one
- If not, use NULL if no NOT NULL clause following the attribute definition
- Otherwise, error

```
Works_ON(ESSN, PNO, Hours)

INSERT INTO Works_ON(ESSN, PNO)
VALUES(123456789, 40);
```

Insert: Example (FYI)

- Provide as a gift for all loan customers of the Perryridge branch, a $200 savings account. Let the loan number serve as the account number for the new savings account

```
insert into account
    select loan-number, branch-name, 200
    from loan
    where branch-name = 'Perryridge'
insert into depositor
    select customer-name, loan-number
    from loan, borrower
    where branch-name = 'Perryridge'
          and loan.account-number = borrower.account-number
```

Set of tuples to insert

- The **select from where** statement is fully evaluated before any of its results are inserted into the relation, *why?*
- What would happen with the following query?

  insert into *table1* **select** * **from** *table1*

Infinite loop!

Update

- Choose tuples to be updated using a query

update R

set *attribute* = expression

where <any construct allowed in where clause>

- Pay 5% interest on accounts whose balance is greater than average

```
UPDATE account
SET balance = balance * 1.05
WHERE balance > any (SELECT AVG(balance)
                     FROM account)
```

Conditional Update (FYI)

- Same query as before: Increase all accounts with balances over $10,000 by 6%, all other accounts receive 5%.

```
UPDATE      Account
SET         balance = 1.05 * balance
WHERE       balance <= 10,000
```

```
UPDATE      Account
SET         balance = 1.06 * balance
WHERE       balance > 10,000
```

SQL Exercise

- SQLite does not allow < ANY. Modify the query below so that it will work in SQLite

```
/* Given a 5% raise to employees with below-average salary */
UPDATE      Employee
SET         Salary = Salary * 1.05
WHERE       Salary < ANY
                    (SELECT     AVG(Salary)
                    FROM        Employee)
```

Database Modification: Exercise

- Give a 10% raise to employee who works on more than 2 projects or whose total project hour exceeds the average (the average of total project hour per employee)
  - E1: 10 hours in total on P1, P2
  - E2: 20 hours in total on P10, P1
  - Average total hour: 15 hours

Database Modification: Exercise - 1

- Give a 10% raise to employee who works on more than 2 projects or whose total project hour exceeds the average (the average of total project hour per employee)

```
UPDATE      Employee
SET         Salary = Salary 1.1
WHERE       SSN IN (
```

Subquery1

UNION

Subquery2

);

Database Modification: Exercise - 2

- Give a 10% raise to employee who works on more than 2 projects or whose total project hour exceeds the average (the average of total project hour per employee)
  - Employee who works on more than 2 projects

```
/* Subquery1 */
SELECT      ESSN
FROM        Works_ON
GROUP BY    ESSN
HAVING      COUNT(PNO) > 2
```

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

Database Modification: Exercise - 3

- Give a 10% raise to employee who works on more than 2 projects or whose total project hour exceeds the average (the average of total project hour per employee)
  - whose total project hour exceeds the average
    - Total project hour for each employee

```
/* Subquery2 */
SELECT      ESSN,
            SUM(Hours) AS THours
FROM        Works_ON
GROUP BY    ESSN
```

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

Database Modification: Exercise - 4

- Give a 10% raise to employee who works on more than 2 projects or whose total project hour exceeds the average (the average of total project hour per employee)
  - whose total project hour exceeds the average
    - Average total project hour

```
/* Subquery2 */
[redacted]
[redacted]      (SELECT     ESSN,
                            SUM(Hours) AS THours
                FROM        Works_ON
                GROUP BY    ESSN)
```

ESSN	THours
123456789	42.0
222222200	40
222222201	48
222222202	40
222222203	40
222222204	40
222222205	40
…	…

Database Modification: Exercise - 5

- Give a 10% raise to employee who works on more than 2 projects or whose total project hour exceeds the average (the average of total project hour per employee)
  - whose total project hour exceeds the average

```
/* Subquery2 */
SELECT   ESSN
FROM     (SELECT   ESSN,
                   SUM(Hours) AS THours
          FROM     Works_ON
          GROUP BY ESSN),

         (SELECT   avg(THours) AS ATH
          FROM     (SELECT   ESSN,
                             SUM(Hours) AS THours
                    FROM     Works_ON
                    GROUP BY ESSN))
WHERE    THours > ATH
```

ESSN	THours
123456789	42.0
222222200	40
222222201	48
222222202	40
222222203	40
222222204	40
222222205	40
…	…

ATH
40.67

Database Modification: Exercise - 6

- Give a 10% raise to employee who works on more than 2 projects or whose total project hour exceeds the average (the average of total project hour per employee)
  - Put the pieces together

Summary of Today's Lecture

- Data definition
  - Create, remove and change schema
- Data modification
  - Create, remove and change an instance