CPSC 304 – Administrative notes November 1 and 5, 2024

- Project:
  - Make sure you have your end-to-end project tech stack working now!!!
  - Milestone 4: Project implementation – due November 29
    - You cannot change your code after this point!
  - Milestone 5: Group demo – week of December 2
  - Milestone 6: Individual Assessment – Due November 29
- Tutorials: basically project group work time
- Final exam: December 16 at 12pm!
  - Conflict form on Piazza – due November 12
- Fall break! No Tutorials/Classes/Office hours November 11-13

Now where were we...

- We'd covered the basics of SQL querying
- What's left for today is SQL potpourri & some in class exercises

What are views

- Relations that are defined with a create table statement exist in the physical layer
  - do not change unless explicitly told so
- Virtual views do not physically exist, they are defined by expression over the tables.
  - Can be queries (most of the time) as if they were tables.

Why use views?

- Hide some data from users
- Make some queries easier
- Modularity of database
  - When not specified exactly based on tables.

Example: UBC has one table for students. Should the CS Department be able to update CS students info? Yes, Biology students? NO
Create a view for CS to only be able to update CS students

Defining and using Views

- Create View <view name><attributes in view>
  As <view definition>
  - View definition is defined in SQL
  - From now on we can use the view almost as if it is just a normal table
- View V (R_1,…R_n)
- query Q involving V
  - Conceptually
    - V (R_1,…R_n) is used to evaluate Q
  - In reality
    - The evaluation is performed over R_1,…R_n

Defining and using Views

- Example: Suppose tables
  Course(Course#,title,dept)
  Enrolled(Course#,sid,mark)

```
CREATE  VIEW  CourseWithFails(dept, course#, mark) AS
      SELECT   C.dept, C.course#, mark
      FROM     Course C, Enrolled E
      WHERE    C.course# = E.course# AND mark<50
```

This view gives the dept, course#, and marks for those courses where someone failed

Views and Security

- Views can be used to present necessary information (or a summary), while hiding details in underlying relation(s).
  - Given CourseWithFails, but not Course or Enrolled, we can find the course in which some students failed, but we can't find the students who failed.

Course(Course#,title,dept)
Enrolled(Course#,sid,mark)
VIEW CourseWithFails(dept, course#, mark)

View Updates

- View updates must occur at the base tables.
  - Ambiguous
  - Difficult

- Example:

```
CREATE  VIEW  CourseWithFails(dept, course#, mark) AS
              SELECT   C.dept, C.course#, mark
              FROM     Course C, Enrolled E
              WHERE    C.course# = E.course# AND mark<50
```

- If you tried to delete a row from CourseWithFails, what does that mean? Do you want to delete the course info or the student info? How can you delete it such that other rows won't be affected?
- DBMS's restrict view updates only to some simple views on single tables (called updatable views)

View Deletes

- Drop View <view name>
  - Dropping a view does not affect any tuples of the in the underlying relation.

- How to handle DROP TABLE if there's a view on the table?
- DROP TABLE command has options to prevent a table from being dropped if views are defined on it:
  - DROP TABLE Student RESTRICT
    - drops the table, unless there is a view on it
  - DROP TABLE Student CASCADE
    - drops the table, and recursively drops any view referencing it

The Beauty of Views

Find those majors for which their average age is the minimum over all majors

With views:

```
Create View Temp(major, average) as
        SELECT      S.major, AVG(S.age) AS average
        FROM        Student S
        GROUP BY  S.major;

SELECT major, average
FROM Temp
WHERE average = (SELECT MIN(average) FROM Temp)
```

Without views:

```
   SELECT  Temp.major, Temp.average
FROM(SELECT  S.major, AVG(S.age) as average
      FROM  Student S
      GROUP BY S.major) AS Temp
WHERE Temp.average in (SELECT  MIN(Temp.average) FROM  Temp)
```

A bit ugly

Clicker question: views

Suppose relation R(a,b,c):

Define the view *V* by:

```
CREATE VIEW V AS
SELECT a+b AS d, c
FROM R;
```

What is the result of the query:

```
SELECT d, SUM(c)
FROM V
GROUP BY d
HAVING COUNT(*) <> 1;
```

a	b	c
1	1	3
1	2	3
2	1	4
2	3	5
2	4	1
3	2	4
3	3	6

Identify, from the list below, a tuple in the result of the query:

A. (2,3)
B. (3,12)
C. (5,9)
D. All are correct
E. None are correct

Clickerview.sql

Clicker question: views

Suppose relation R(a,b,c):

Define the view *V* by:

```
CREATE VIEW V AS
SELECT a+b AS d, c
FROM R;
```

What is the result of the query:

```
SELECT d, SUM(c)
FROM V
GROUP BY d
HAVING COUNT(*) <> 1;
```

a	b	c
1	1	3
1	2	3
2	1	4
2	3	5
2	4	1
3	2	4
3	3	6

V

d	c
2	3
3	3
3	4
5	5
6	1
5	4
6	6

d	Sum(C)
3	7
5	9
6	7

Identify, from the list below, a tuple in the result of the query:

A. (2,3) Wrong. In view
B. (3,12)
C. (5,9) Right
D. All are correct
E. None are correct

Null Values

- Tuples may have a null value, denoted by *null*, for some of their attributes
- Value *null* signifies an unknown value or that a value does not exist.
- The predicate **IS NULL** (**IS NOT NULL**) can be used to check for null values.
  - E.g. *Find all student names whose age is not known.*

```
SELECT name
FROM Student
WHERE age IS NULL
```

- The result of any arithmetic expression involving *null* is *null*
  - E.g. 5 + *null* returns *null.*

Null Values and Three Valued Logic

- null requires a 3-valued logic using the truth value *unknown*:
  - OR: (*unknown* **or** *true*) = *true*, (*unknown* **or** *false*) = *unknown*
    (*unknown* **or** *unknown*) = *unknown*
  - AND: *(true* **and** *unknown)* = *unknown, (false* **and** *unknown)* = *false,*
    (unknown **and** *unknown)* = *unknown*
  - NOT*: (***not** *unknown)* = *unknown*
  - "*P* **is unknown"** evaluates to true if predicate *P* evaluates to *unknown*
- Any comparison with *null* returns *unknown*
  - *E.g. 5 < null or null <> null or null = null*
- Result of **where** clause predicate is treated as *false* if it evaluates to *unknown*
- All aggregate operations except **count(*)** ignore tuples with null values on the aggregated attributes.

SELECT count(*)
FROM class

SELECT count(fid)
FROM class

Clicker null query

Determine the result of:

```
SELECT COUNT(*),
    COUNT(Runs)
FROM Scores
WHERE Team = 'Carp'
```

Which of the following is in the result:

A. (1,0)
B. (2,0)
C. (1,NULL)
D. All of the above
E. None of the above

Scores:

Team	Day	Opponent	Runs
Dragons	Sun	Swallows	4
Tigers	Sun	Bay Stars	9
Carp	Sun	NULL	NULL
Swallows	Sun	Dragons	7
Bay Stars	Sun	Tigers	2
Giants	Sun	NULL	NULL
Dragons	Mon	Carp	NULL
Tigers	Mon	NULL	NULL
Carp	Mon	Dragons	NULL
Swallows	Mon	Giants	0
Bay Stars	Mon	NULL	NULL
Giants	Mon	Swallows	5

Clicker null query

Start clickernull.sql

Determine the result of:

```
SELECT COUNT(*),
    COUNT(Runs)
FROM Scores
WHERE Team = 'Carp'
```

Which of the following is in the result:

A. (1,0)
B. (2,0) Right
C. (1,NULL)
D. All of the above
E. None of the above

Scores:

Team	Day	Opponent	Runs
Dragons	Sun	Swallows	4
Tigers	Sun	Bay Stars	9
Carp	Sun	NULL	NULL
Swallows	Sun	Dragons	7
Bay Stars	Sun	Tigers	2
Giants	Sun	NULL	NULL
Dragons	Mon	Carp	NULL
Tigers	Mon	NULL	NULL
Carp	Mon	Dragons	NULL
Swallows	Mon	Giants	0
Bay Stars	Mon	NULL	NULL
Giants	Mon	Swallows	5

Previously…

```
SELECT DISTINCT cname
FROM Enrolled e, Student s
WHERE e.snum = s.snum
```

Natural Join

- The SQL NATURAL JOIN is a type of EQUI JOIN and is structured in such a way that, columns with same name of associate tables will appear once only.
- Natural Join : Guidelines
  - The associated tables have one or more pairs of identically named columns.
  - The columns must be the same data type.
  - Don’t use ON clause (similar to where, but just joins) in a natural join.

```
SELECT *
FROM student s natural join enrolled e
```

- Natural join of tables with no pairs of identically named columns will return the cross product of the two tables.

```
SELECT *
FROM student s natural join class c
```

More fun with joins

- What happens if I execute query:

```
SELECT *
FROM student s, enrolled e
WHERE s.snum = e.snum
```

- To get *all* students, you need an *outer join*
- There are several special joins declared in the *FROM* clause:
  - Inner join – default: only include matches
  - Left outer join – include all tuples from left hand relation
  - Right outer join – include all tuples from right hand relation
  - Full outer join – include all tuples from both relations
- Orthogonal: can have natural join (as in relational algebra)

Example: SELECT *

FROM Student S NATURAL LEFT OUTER JOIN Enrolled E

More fun with joins examples

R

A	B
1	2
3	3

S

B	C
2	4
4	6

Natural Inner Join

A	B	C
1	2	4

Natural Left outer Join

A	B	C
1	2	4
3	3	Null

Natural Right outer Join

A	B	C
1	2	4
Null	4	6

Natural Full outer Join

A	B	C
1	2	4
3	3	Null
Null	4	6

Outer join (without the Natural) will use the key word ON for specifying The condition of the join.

Outer join not implemented in MYSQL
Outer join is implemented in Oracle

Clicker outer join question

- Given:
  Compute:
  SELECT R.A, R.B, S.B, S.C, S.D
  FROM R FULL OUTER JOIN S
  ON (R.A > S.B AND R.B = S.C)
- Which of the following tuples of R or S is dangling (and therefore needs to be padded in the outer join)?

A. (1,2) of R
B. (3,4) of R
C. (2,4,6) of S
D. All of the above
E. None of the above

R(A,B) S(B,C,D)

A	B	B	C	D
1	2	2	4	6
3	4	4	6	8
5	6	4	7	9

Start clickerouter.sql

Clicker outer join question

- Given:
  Compute:
  SELECT R.A, R.B, S.B, S.C, S.D
  FROM R FULL OUTER JOIN S
  ON (R.A > S.B AND R.B = S.C)
- Which of the following tuples of R or S is dangling (and therefore needs to be padded in the outer join)?

A. (1,2) of R
B. (3,4) of R
C. (2,4,6) of S
D. All of the above
E. None of the above

A is correct

R(A,B) S(B,C,D)

A	B
1	2
3	4
5	6

B	C	D
2	4	6
4	6	8
4	7	9

A	B	B	C	D
3	4	2	4	6
5	6	4	6	8
1	2	NULL	NULL	NULL
NULL	NULL	4	7	9

Database Manipulation
Insertion redux

- Can insert a single tuple using:

```
INSERT  INTO  Student
VALUES  (53688, 'Smith', '222 W.15th ave', 333-4444, MATH)
```

or

```
INSERT  INTO  Student (sid, name, address, phone, major)
VALUES  (53688, 'Smith', '222 W.15th ave', 333-4444, MATH)
```

- Add a tuple to student with null address and phone:

```
INSERT  INTO  Student (sid, name, address, phone, major)
VALUES  (33388, 'Chan', null, null, CPSC)
```

Database Manipulation
Insertion redux (cont)

- Can add values selected from another table
- Enroll student 51135593 into every class taught by faculty 90873519

```
INSERT  INTO  Enrolled
SELECT  51135593, name
FROM Class
WHERE fid = 90873519
```

The SELECT-FROM-WHERE statement is fully evaluated before any of its results are inserted or deleted.

Database Manipulation
Deletion

- Note that only whole tuples are deleted.
- Can delete all tuples satisfying some condition (e.g., name = Smith):

```
DELETE   FROM  Student
WHERE   name = 'Smith'
```

- The WHERE clause can contain nested queries

Database Manipulation Updates

- Increase the age of all students by 2 (should not be more than 100)
- Need to write two updates:

```
UPDATE  Student
SET     age = 100
WHERE   age >= 98

UPDATE  Student
SET age = age + 2
WHERE age < 98
```

Database Manipulation Updates

- Increase the age of all students by 2 (should not be more than 100)
- Need to write two updates:

```
UPDATE  Student
SET     age = 100
WHERE   age >= 98

UPDATE  Student
SET age = age + 2
WHERE age < 98
```

- Is the order important?

A: Yes B: No

Integrity Constraints (Review)

- An IC describes conditions that every *legal instance* of a relation must satisfy.
  - Inserts/deletes/updates that violate IC's are disallowed.
  - Can ensure application semantics (e.g., *sid* is a key), or prevent inconsistencies (e.g., *sname* has to be a string, *age* must be < 200)
- *Types of IC's*:
  - domain constraints,
  - primary key constraints,
  - foreign key constraints,
  - general constraints

General Constraints: Check

- We can specify constraints over a single table using table constraints, which have the form

Check conditional-expression

```
CREATE TABLE  Student
   ( snum  INTEGER,
     sname  CHAR(32),
     major  CHAR(32),
     standing CHAR(2)
     age  REAL,
     PRIMARY KEY  (snum),
     CHECK  ( age >= 10
              AND age < 100 );
```

Check constraints are checked when tuples are inserted or modified

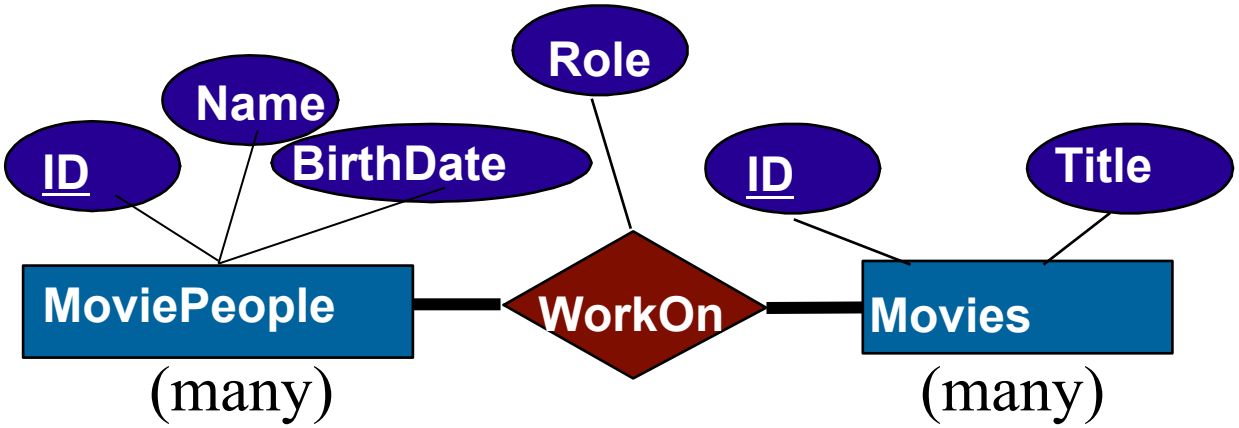
General Constraints: Check

- Constraints can be named
- Can use subqueries to express constraint
- Table constraints are associated with a single table, although the conditional expression in the check clause can refer to other tables

```
CREATE TABLE Enrolled
    ( snum    INTEGER,
      cname   CHAR(32),
      PRIMARY KEY  (snum, cname),
      CONSTRAINT noR15
      CHECK  (`R15' <>
                ( SELECT  c.room
                  FROM     class c
                  WHERE  c.name=cname)));
```

No one can be enrolled in a class, which is held in R15

Constraints over Multiple Relations: Remember this one?



- We couldn't express "every MoviePerson works on some movie and every movie has some MoviePeople working on it"?
- Neither foreign-key nor not-null constraints in Works_In can do that.
- Assertions to the rescue!

Constraints Over Multiple Relations

- Cannot be defined in one table.
- Are defined as ASSERTIONs which are not associated with any table
- Example: *Every MovieStar needs to star in at least one Movie*

```
CREATE ASSERTION  totalEmployment
CHECK
( NOT EXISTS ((SELECT StarID FROM MovieStar)
                    EXCEPT
                   (SELECT StarID FROM StarsIn)));
```

Constraints Over Multiple Relations

- Example: Write an assertion to enforce every student to be registered in at least one course.
  Student(snum, sname, major, standing, age)
  Enrolled(snum, cname)
  Class(name, meets_at, fid)

```
CREATE ASSERTION  Checkregistry
CHECK
( NOT EXISTS ((SELECT snum FROM student)
                    EXCEPT
                   (SELECT snum FROM enrolled)));
```

Triggers

- Trigger : a procedure that starts automatically if specified changes occur to the DBMS
- Active Database: a database with triggers
- A trigger has three parts:
  1. Event (activates the trigger)
  2. Condition (tests whether the trigger should run)
  3. Action (procedure executed when trigger runs)

Useful for project
Not tested on exams

- Database vendors did not wait for trigger standards! So trigger format depends on the DBMS
- NOTE: triggers may cause cascading effects.

Good way to shoot yourself in the foot

Triggers: Example (SQL:1999)

```
CREATE TRIGGER youngStudentUpdate
   AFTER INSERT ON Student
REFERENCING NEW TABLE NewStudent
FOR EACH STATEMENT
   INSERT INTO
      YoungStudent(snum, sname, major, standing, age)
      SELECT  snum, sname, major, standing, age
      FROM    NewStudent N
      WHERE  N.age <= 18;
```

event

newly inserted tuples

apply once per statement

action

Can be either before or after

That's nice. But how do we code with SQL?

- Direct SQL is rarely used: usually, SQL is embedded in some application code.
- We need some method to reference SQL statements.
- But: there is an *impedance mismatch* problem.
  - Structures in databases <> structures in programming languages
- Many things can be explained with the impedance mismatch.

The Impedance Mismatch Problem

The host language manipulates variables, values, pointers SQL manipulates relations.

There is no construct in the host language for manipulating relations. See https://en.wikipedia.org/wiki/Object-relational_impedance_mismatch

Why not use only one language?

- Forgetting SQL: “we can quickly dispense with this idea” [Ullman & Widom, pg. 363].
- SQL cannot do everything that the host language can do.

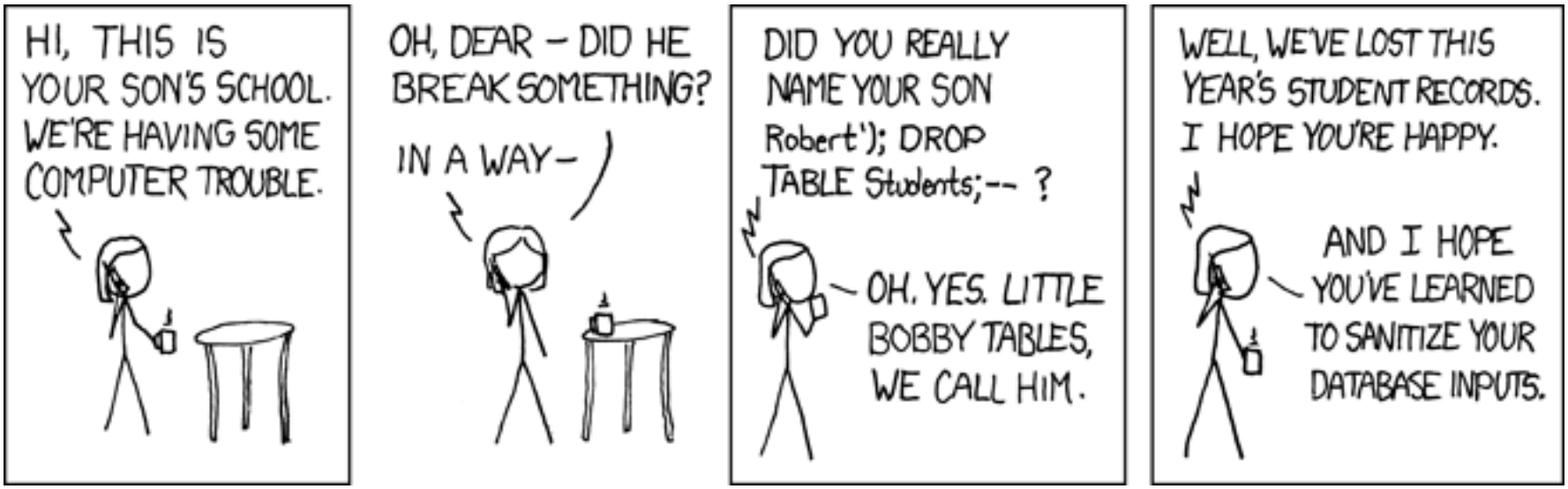
Database APIs

Rather than modify compiler, add library with database calls (API)

- Special standardized interface: procedures/objects
- Passes SQL strings from language, presents result sets in a language-friendly way – solves that impedance mismatch
- Microsoft's *ODBC* is a C/C++ standard on Windows
- Sun's *JDBC* a Java equivalent
- API's are DBMS-neutral
  - a "driver" traps the calls and translates them into DBMS-specific code

And now a brief digression

- Have you ever wondered why some websites don't allow special characters?



Summary

- SQL was an important factor in the early acceptance of the relational model; more natural than earlier, procedural query languages.
- Relationally complete; in fact, significantly more expressive power than relational algebra.
- Consists of a data definition, data manipulation and query language.
- Many alternative ways to write a query; optimizer should look for most efficient evaluation plan.
  - In practice, users need to be aware of how queries are optimized and evaluated for best results.

Summary (Cont')

- NULL for unknown field values brings many complications
- SQL allows specification of rich integrity constraints (and triggers)
- Embedded SQL allows execution within a host language; cursor mechanism allows retrieval of one record at a time
- APIs such as ODBC and JDBC introduce a layer of abstraction between application and DBMS

Learning Goals Revisited

- Given the schemas of a relation, create SQL queries using: SELECT, FROM, WHERE, EXISTS, NOT EXISTS, UNIQUE, NOT UNIQUE, ANY, ALL, DISTINCT, GROUP BY and HAVING.
- Show that there are alternative ways of coding SQL queries to yield the same result. Determine whether or not two SQL queries are equivalent.
- Given a SQL query and table schemas and instances, compute the query result.
- Translate a query between SQL and RA.
- Comment on the relative expressive power of SQL and RA.
- Explain the purpose of NULL values and justify their use. Also describe the difficulties added by having nulls.
- Create and modify table schemas and views in SQL.
- Explain the role and advantages of embedding SQL in application programs.
- Write SQL for a small-to-medium sized programming application that requires database access.
- Identify the pros and cons of using general table constraints (e.g., CONSTRAINT, CHECK) and triggers in databases.

In class exercise

- SQL 4
- SQL 3 is not for credit, but do it anyway because it's good for you