Practice: Hotel Schema

Hotel

hotelNumber	hotelName	city

Room

roomNumber	hotelNumber	type	price

Guest

guestNumber	guestName	guestAddress

Booking

hotelNumber	guestNumber	dateFrom	dateTo	roomNumber

Practice: Hotel Schema

- List all guests currently staying at the Winston hotel.
  - RA: Guest $\triangleright$ $_{\text{Guest.guestNumber=Booking.GuestNumber}}$ ($\sigma_{\text{dateFrom} \leq \text{'02-09-12'} \;\&\&\; \text{dateTo} \geq \text{'02-09-12'}}$ (Booking $\triangleright$ $\triangleleft_{\text{Booking.hotelNumber=Hotel.hotelNumber}}$ ($\sigma_{\text{hotelName} = \text{'Winston'}}$(Hotel))))

```
SELECT g.guestName FROM guest g NATURAL JOIN (SELECT * FROM booking b
NATURAL JOIN (SELECT * FROM hotel WHERE hotelName='Winston Hotel') as w
WHERE dateFrom <= '2012-02-16' AND dateTo >= '2012-02-16') AS z;
```

- List the details of all rooms at the Winston hotel, including the name of the guest staying in the room if the room is occupied.
  - RA: (Room $\triangleright\triangleleft_{\text{Room.hotelNumber=Hotel.hotelNumber}}$ ($\sigma_{\text{hotelName} = \text{'Winston'}}$(Hotel)) LOJ $_{\text{roomNumber,hotelNumber}}$ ($\Pi_{\text{guestName,roomNumber,hotelNumber}}$(Guest $\triangleright\triangleleft$ $\sigma_{\text{dateFrom} \leq \text{'02-09-12'} \;\&\&\; \text{dateTo} \geq \text{'02-09-12'}}$ (Booking $\triangleright\triangleleft$ ($\sigma_{\text{hotelName} = \text{'Winston'}}$(Hotel))))

```
SELECT n.*,o.guestName FROM (SELECT * FROM room NATURAL JOIN (SELECT
* FROM hotel WHERE hotelName='Winston Hotel') AS m) AS n LEFT JOIN (SELECT
guestName,roomNumber,hotelNumber FROM guest NATURAL JOIN (SELECT *
FROM booking b NATURAL JOIN (SELECT * FROM hotel WHERE
hotelName='Winston Hotel') as w WHERE dateFrom <= '2012-02-16' AND dateTo >=
'2012-02-16') as z) AS o ON n.roomNumber=o.roomNumber AND
n.hotelNumber=o.hotelNumber;
```

SQL: Select

Another SELECT variation: EXISTS / NOT EXISTS

Employed in WHERE clauses with sub-queries

EXISTS returns true if the returned table from the subquery has at least one row

NOT EXISTS does the opposite (returns true if the returned table is empty)

SQL: Select

```
SELECT * FROM Course c WHERE EXISTS (SELECT
* FROM Enrollment e WHERE  c.courseID =
e.courseID)
```

What does this get us?

All courses with actual enrollment

SQL: Combining Results Tables

Given tables A and B,

- A UNION B (Union)
  - Table containing all rows that are in A or B or both
- A INTERSECT B (Intersection)
  - Table containing all rows that are in A and B
- A EXCEPT B (Difference)
  - Table containing all rows that are in A and not in B

Union Example

- Another poor example, but gets the syntactical idea across for student domain:

SELECT lastName, firstName FROM student WHERE major='CSC' UNION SELECT lastName,firstName FROM student WHERE major='MTH';

Returns all CSC and MTH majors
This example returns same table as:
SELECT lastName,firstName FROM student
WHERE major='CSC' OR major='MTH';

COMBINING TABLES

These two provide opposite answers… what do they tell us?

(SELECT courseID FROM Course) EXCEPT (SELECT courseID FROM Enrollment)

courseIDs for courses with zero enrollment

(SELECT courseID FROM Course) INTERSECT (SELECT courseID FROM Enrollment)

courseIDs for courses with non-zero enrollment

SQL: Combining Results Tables

- Remember that the tables have to be “union-compatible”
  - Are they in my examples?

  (SELECT courseID FROM Course) EXCEPT (SELECT courseID FROM Enrollment)

- Unfortunately MySQL doesn’t support INTERSECT and EXCEPT, nor DIVIDE!!

SQL: Workarounds

- Can we fashion our own Workarounds that achieve the same semantics as INTERSECT, EXCEPT, and DIVIDE?

- Yes
  - Useful to have a notion of row constructor first

SQL: Row Constructor

Think back to INSERT statement

INSERT INTO tableName VALUES (, , ,);

The (, , ,) essentially defines a tuple that is being generated to be stored into the table called tableName

One can construct “rows” on the fly and compare rows:

SELECT * FROM t1 WHERE (col1,col2) = (SELECT col3, col4 FROM t2 WHERE id = 10);

Row Constructors

SELECT * FROM STUDENT WHERE (major, GPA) = (‘CSC’,3.5);

is semantically equivalent to

SELECT * FROM STUDENT WHERE major = ‘CSC’ AND GPA = 3.5;

MySQL: INTERSECT Workaround

Want all rows that are common between two tables with the same structure. Assume our tables just have attributes *x,y*

SELECT * FROM a WHERE (x,y) IN (SELECT * FROM b)

SELECT * FROM a WHERE EXISTS (SELECT * FROM b WHERE b.x = a.x AND b.y = a.y); // this one doesn't require the same order in the tables!

SELECT DISTINCT a.x, a.y FROM a JOIN b USING (x,y);

USING (x,y) is equivalent to where b.x = a.x AND b.y = a.y
(says what columns to use in the join filter , assumes equality test)

Should these all work in providing the same results as an INTERSECT?
If, so, should we prefer one over the others?
For all, we are enumerating the attributes at some point

SQL EXPLAIN

- The EXPLAIN command in SQL can describe what tables are generated during execution of an operation
  - Can help us to understand preferences for one query style over another.
  - Syntax: EXPLAIN queryGoesHere

MySQL: EXCEPT Workaround

Want all rows that are unique to the LHS table (table A) of A EXCEPT B

SELECT * FROM a WHERE (x,y) NOT IN (SELECT * FROM b)

SELECT * FROM a WHERE NOT EXISTS (SELECT * FROM b WHERE b.x = a.x AND b.y = a.y);

SELECT DISTINCT a.x, a.y FROM a LEFT JOIN b USING (x,y) where b.x IS NULL ;

Should these all work in providing the same results as an INTERSECT?
If, so, should we prefer one over the others?
For all, we are enumerating the attributes at some point

Semantically Equivalent Examples

Except: (SELECT courseID FROM Course) EXCEPT (SELECT courseID FROM Enrollment)←→

SELECT DISTINCT course.courseID FROM course LEFT JOIN enrollment USING (courseID) WHERE enrollment.courseID IS NULL;

Intersect: (SELECT courseID FROM Course) INTERSECT (SELECT courseID FROM Enrollment) →

SELECT DISTINCT course.courseID FROM course JOIN enrollment USING (courseID);

MySQL: DIVIDE Workaround

- DIVIDE: Pull out instances in table X affiliated with all instances in table Y
  - Suppliers of all parts
  - Students in all classes

SELECT DISTINCT x FROM a a1 WHERE NOT EXISTS (SELECT y FROM b WHERE y NOT IN (SELECT y FROM a a2 WHERE a1.y = a2.y));

Courses with every student in it: enrollment / studentID

SELECT DISTINCT courseID FROM enrollment e WHERE NOT EXISTS (SELECT studentID FROM student WHERE studentID NOT IN (SELECT studentID FROM enrollment e2 WHERE e2.courseID = e.courseID);

SQL: Additional Data Manipulation

- Have seen
  - Query: SELECT x FROM y WHERE z;
  - Insert: INSERT INTO x VALUES(a,b,c);
- Can also
  - Modify
  - Delete

SQL: Modifying Data

- Modifying values is essentially a combination of INSERT and SELECT

UPDATE TableName SET columnName1 = value1 [,columnName2=value2,…] [WHERE searchCondition]

If no WHERE is supplied applies to the whole table; otherwise, only applies to rows passing filter

SQL: UPDATE Example

UPDATE student SET major='CSCS' WHERE major='CSC';

All CSC majors are turned into CSCS majors

Needed to be applied when we first introduced the BA in CS

SQL: Deleting Data

- Deleting values is essentially an analogue of SELECT

DELETE FROM TableName WHERE searchCondition

If no WHERE is supplied applies to the whole table; otherwise, only applies to rows passing filter

SQL: DELETE Example

DELETE FROM enrollment WHERE courseID='06902';

Remove all enrollment information for a particular class

Useful when a class is dropped from the schedule

Assignment

- A new assignment asking you to practice queries was passed out. Please work on it (due Wednesday) and we will spend some time on it in class Tuesday.

Project

- Information and a schedule for the course project was presented. Also available in Sakai.