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

- Generate correct (and reasonable, given assumptions you may need to make about the intent of the query) relational algebra expressions for the following queries:
  - List all guests currently staying at the Winston hotel.
  - 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.
  - List the guest details of all guests staying at the Winston hotel.

Practice: Hotel Schema

- List all hotels.

  RA: Hotel

- List all single rooms with a price below $100 a night.

  RA: $\sigma_{type='single' \ \&\& \ price<\$100}(Room)$

- List the names and cities of all guests.

  RA: $\Pi_{guestName,guestNumber}(Guest)$

- List the price and type of all rooms at the Winston hotel.

  RA: $\Pi_{price,type}(Room \triangleright_{Room.hotelNumber=Hotel.hotelNumber} (\sigma_{hotelName='Winston'}(Hotel)))$

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 $\triangleright\triangleleft_{\text{Booking.hotelNumber=Hotel.hotelNumber}}$ ($\sigma_{\text{hotelName = 'Winston'}}$(Hotel))))

- 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 $\triangleright\triangleleft_{\text{Room.hotelNumber=Hotel.hotelNumber}}$ ($\sigma_{\text{hotelName = 'Winston'}}$(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 $\triangleright\triangleleft$ ($\sigma_{\text{hotelName = 'Winston'}}$(Hotel))))

- List the guest details of all guests staying at the Winston hotel.

  RA: Guest $\triangleright$ (Booking $\triangleright\triangleleft$ ($\sigma_{\text{hotelName = 'Winston'}}$(Hotel)))

SQL: Data Manipulation Language

- SQL:
  - DDL: Data Definition Language
    - Creating , modifying, and deleting tables
  - DML: Data Manipulation Language
    - Inserting data into, extracting data from, deleting data from, and updating data in tables
  - SQL *select* statement -- the workhorse

SQL SELECT

- Focus on simple uses of SELECT first that map to simple relational operations
- Key idea: Specify *what you want*, not *how to get it*
  - *Non-procedural*
- *Note:* We will look at SQL-standard syntax – may not be matched precisely or fully supported by MYSQL

SQL SELECT

Projection

```
SELECT [DISTINCT | ALL]
    {* | [columnExpression [AS newName]] [,...] }
FROM        TableName [alias]
[WHERE  condition]
```

Selection

SQL SELECT

```
SELECT [DISTINCT | ALL]
   {* | [columnExpression [AS newName]] [,...] }
FROM   TableName [alias]
[WHERE     condition]
```

{* | [columnExpression [AS newName]] [,...] }

- return all columns (*), or return only columns provided in comma separated list (renamed if desired)

DISTINCT – drop repeated rows

[alias] – nickname we can assign table

[WHERE condition] – a boolean expression over attributes

Example Relations

Student

studentID	lastName	firstName	year	major	GPA
1123	Smith	Robert	4	CSC	3.5
1129	Jones	Douglas	3	MTH	2.9
1145	Brady	Susan	4	CSC	3.8

Course

courseID	dept.	number
06902	CSC	221
06903	CSC	231
06904	CSC	241
06905	CSC	191

Enrollment

studentID	courseID
1123	06902
1129	06902
1145	06902
1123	06903
1145	06903
1123	06904
1129	06904

SQL SELECT: Examples

- SELECT * FROM Student:

Student

studentID	lastName	firstName	year	major	GPA
1123	Smith	Robert	4	CSC	3.5
1129	Jones	Douglas	3	MTH	2.9
1145	Brady	Susan	4	CSC	3.8

- SELECT DISTINCT major FROM Student:

major
CSC
MTH

- SELECT lastName, firstName FROM Student WHERE GPA > 3.0

lastName	firstName
Smith	Robert
Brady	Susan

SQL SELECT

Textbook specifies variations on the WHERE statement that SQL supports, including:

Compound comparisons (X BOOL Y AND A BOOL B)

Range comparisons (X BETWEEN Y AND Z)

Set membership (X IN (Enumeration,…))

NULL (X IS/IS NOT NULL)

Like Pattern matching:

% = wildcard (zero or more characters)
_ = one character

SELECT * FROM Student WHERE major LIKE ‘CSC_’
Get all WFU CSC majors (CSCS, CSCA)

SELECT * FROM Student WHERE lastName LIKE ‘T%’
Get all WFU students whose last name starts with T

SQL: Sorting Results

Query results don't normally correspond to any particular ordering

Results can be sorted by columns, using ORDER BY option

An arbitrary number of columns can be listed, will be sorted by first named column first, then down the sequence

Can request ASCending or DESCending on each column

SELECT lastName, firstName, GPA FROM Student WHERE GPA > 3.0 ORDER BY GPA DESC;

lastName	firstName	GPA
Brady	Susan	3.8
Smith	Robert	3.5

SQL: Aggregate Functions

SQL can apply aggregate functions to columns in tables resulting from queries, providing summary information

Available in SQL standard: COUNT (and COUNT(*)), SUM, MIN, MAX, AVG

Others may also be defined in local implementations

- One place can show up in SELECT list part of statement
- Generally, throws away NULLs, except COUNT(*)
- Can request to only count DISTINCT rows with COUNT(DISTINCT field) syntax
- If don't use GROUP BY mechanism (next few slides), then since this is generating summary information, can't select other columns not involved in the aggregate function unless applying an aggregate function to those columns

SQL: Aggregate Functions

SELECT COUNT(*) FROM Student;

➔ *3*

SELECT COUNT(*) FROM Student WHERE GPA > 3.0;

➔ *2*

SELECT COUNT(DISTINCT major) from STUDENT;

➔ *2*

SELECT COUNT(DISTINCT major) from STUDENT where GPA > 3.0;

➔ *1*

SELECT MAX(GPA) from STUDENT

➔ *3.8*

SELECT AVG(GPA) from STUDENT

➔ *3.4*

SELECT COUNT(studentID), AVG(GPA) FROM Student WHERE major='CSC'

➔ *2 3.65*

studentID	lastName	firstName	year	major	GPA
1123	Smith	Robert	4	CSC	3.5
1129	Jones	Douglas	3	MTH	2.9
1145	Brady	Susan	4	CSC	3.8

SQL: Grouping

- Aggregate functions as discussed are applied across *all* rows from the resulting table
- Possible to group, based on shared values in given columns, the rows in the resulting table and present summary information for the groups

- Employs GROUP BY and HAVING clauses

SQL: Grouping

- Aggregate functions as discussed are applied across *all* rows from the resulting table
- Possible to group, based on shared values in given columns, the rows in the resulting table and present summary information for the groups

- Employs GROUP BY and HAVING clauses

SQL: Grouping

- SELECT courseID, COUNT(studentID) AS studentCount FROM Enrollment GROUP BY courseID;

Enrollment

studentID	courseID
1123	06902
1129	06902
1145	06902
1123	06903
1145	06903
1123	06904
1129	06904

studentID	courseID
1123	06902
1129	06902
1145	06902
1123	06903
1145	06903
1123	06904
1129	06904

courseID	studentCount
06902	3
06903	2
06904	2

How is this different from: SELECT (COUNT(*)) FROM Enrollment WHERE courseID = 06902?

SQL: Grouping

- GROUP BY column1 [,column2,…]
- *SELECT expression* must be closely tied to *GROUP BY expression*
  - Each item in SELECT expression must be single-valued for group
  - SELECT can only contain column names, aggregate functions, constants, and expressions combining the previous three elements
  - Grouping happens after selection (WHERE) and before PROJECTION (SELECT)
    - All column names in the SELECT expression must have shown up in the GROUP BY list unless part of aggregate function in SELECT expression

SQL: Having

- WHERE clause implements relational operator *selection*, choosing which rows go into an output table

- A HAVING clause allows one to filter which GROUPS appear in an output table when GROUPING is used
- HAVING clause can only contain columns from GROUP BY clause or aggregate expressions

SQL: Having

- SELECT courseID, COUNT(studentID) AS studentCount FROM Enrollment GROUP BY courseID *HAVING count(studentID) > 2*

Enrollment

studentID	courseID
1123	06902
1129	06902
1145	06902
1123	06903
1145	06903
1123	06904
1129	06904

studentID	courseID
1123	06902
1129	06902
1145	06902
1123	06903
1145	06903
1123	06904
1129	06904

courseID	studentCount
06902	3

SQL: Subqueries

- Possible to nest a query within a query
  - Used in WHERE and HAVING clauses to return values for filter
  - *Scalar:* returns a single value
  - *Row:* returns one row, with multiple columns
  - *Table:* returns multiple rows, with one or more columns
    - *Column:* returns multiple rows, with one column

SQL: Subqueries

SELECT courseID FROM Enrollment WHERE studentID =
(SELECT studentID FROM Student WHERE lastName =
'Smith' AND firstName = 'Robert')

studentID	lastName	firstName	year	major	GPA
1123	Smith	Robert	4	CSC	3.5
1129	Jones	Douglas	3	MTH	2.9
1145	Brady	Susan	4	CSC	3.8

studentID	courseID
1123	06902
1129	06902
1145	06902
1123	06903
1145	06903
1123	06904
1129	06904

Inner query, Outer query

Inner query always in parentheses

courseID
06902
06903
06904

SQL: Subqueries

SELECT lastName, firstName FROM Student WHERE GPA > (SELECT AVG(GPA) AS avgGPA FROM Student)

studentID	lastName	firstName	year	major	GPA
1123	Smith	Robert	4	CSC	3.5
1129	Jones	Douglas	3	MTH	2.9
1145	Brady	Susan	4	CSC	3.8

avgGPA
3.4

lastName	firstName
Smith	Robert
Brady	Susan

SQL: Subqueries

- SQL subquery syntax rules:
  - Can’t use ORDER BY
  - If unqualified, column names refer to the table in the subquery FROM clause
  - Subquery results should always be on the RHS of any Boolean operators in the WHERE/HAVING clause the subquery is in

- There may be ways to write SELECT statements sub-queries that use alternatives (JOINS, MIX, MAX)
  - Let’s keep our eyes open!

SQL: Subqueries

- If sub-query returns multiple rows, can employ further extensions:
  - IN
    - Returns true if value being searched for is in results from sub-query
  - ANY
    - Returns true if condition is satisfied by at least one value produced by the sub-query
  - ALL
    - Returns true if condition is satisfied by all values produced by the sub-query

SQL: Sub-query examples

- *IN:* SELECT dept, number FROM Course WHERE courseID IN (SELECT courseID FROM Enrollment)

  Show dept name and course number for all courses with students enrolled

- *ALL:* SELECT lastName, firstName FROM Student WHERE GPA > ALL (SELECT GPA FROM Student WHERE major = 'CSC')

  Show last name, first name of all students who have higher GPAS than ALL CSC students [~ all students with better GPAs than best CSC student]

What if we had used "ANY" instead – what would that be asking, in English terms?

SQL: Sub-query

- Can use sub-query as component of FROM statement , as long as provide an alias

SELECT … FROM (subquery) AS name

Poor example (but gets idea across):

SELECT studentID FROM (SELECT * FROM student WHERE major =‘CSC’) AS CSCstudent;

More useful:

SELECT DISTINCT courseID FROM enrollment , (SELECT * FROM student WHERE major=‘CSC’) as CSCstudent WHERE enrollment.studentID = CSCstudent.studentID;

This actually uses a join, we’ll come back to it…

SQL: Joins

Employing JOIN syntax allows queries to return attributes from *multiple* tables

Sub-queries did not allow that (they only helped in filtering)

To specify simple joins,

- List tables of interest in FROM clause, comma-separated
- Employ table aliases
- Use WHERE clause constraint to set join constraint, using aliases to disambiguate columns
- ORDER BY and GROUP BY can be employed on the results of joins as appropriate.

SQL: Joins

- SELECT lastName, firstName, courseID FROM Student s, Enrollment e WHERE s.studentID = e.studentID

studentID	lastName	firstName	year	major	GPA
1123	Smith	Robert	4	CSC	3.5
1129	Jones	Douglas	3	MTH	2.9
1145	Brady	Susan	4	CSC	3.8

studentID	courseID
1123	06902
1129	06902
1145	06902
1123	06903
1145	06903
1123	06904
1129	06904

This is an equijoin

lastName	firstName	courseID
Smith	Robert	06902
Jones	Doug	06902
Brady	Susan	06902
Smith	Robert	06903
Brady	Susan	06903
Smith	Robert	06904
Jones	Doug	06904

SQL: Joins (Three Tables)

- SELECT lastName, firstName, dept, number FROM Student s, Course c, Enrollment e WHERE s.studentID = e.studentID AND e.courseID = c.courseID

studentID	lastName	firstName	year	major	GPA
1123	Smith	Robert	4	CSC	3.5
1129	Jones	Douglas	3	MTH	2.9
1145	Brady	Susan	4	CSC	3.8

studentID	courseID
1123	06902
1129	06902
1145	06902
1123	06903
1145	06903
1123	06904
1129	06904

courseID	dept.	number
06902	CSC	221
06903	CSC	231
06904	CSC	241
06905	CSC	191

lastName	firstName	dept	number
Smith	Robert	CSC	221
Jones	Doug	CSC	221
Brady	Susan	CSC	221
Smith	Robert	CSC	231
Brady	Susan	CSC	231
Smith	Robert	CSC	241
Jones	Doug	CSC	241

SQL: Outer Joins

- If we want to preserve tuples from a side of the join, use LEFT, RIGHT, or FULL JOIN
  - SELECT x FROM y LEFT JOIN z ON boolean comparison
  - SELECT x FROM y RIGHT JOIN z ON boolean comparison
  - SELECT x FROM y FULL JOIN z ON boolean comparison

SQL: Outer Joins

- SELECT dept, number, studentID FROM Course c LEFT JOIN Enrollment e ON c.courseID = e.courseID

studentID	courseID
1123	06902
1129	06902
1145	06902
1123	06903
1145	06903
1123	06904
1129	06904

courseID	dept.	number
06902	CSC	221
06903	CSC	231
06904	CSC	241
06905	CSC	191

dept	number	studentID
CSC	221	1123
CSC	221	1129
CSC	221	1145
CSC	231	1123
CSC	231	1145
CSC	241	1123
CSC	241	1129
CSC	191	NULL

SQL: Natural Joins

- SELECT * FROM Course NATURAL JOIN Enrollment

studentID	courseID
1123	06902
1129	06902
1145	06902
1123	06903
1145	06903
1123	06904
1129	06904

courseID	dept.	number
06902	CSC	221
06903	CSC	231
06904	CSC	241
06905	CSC	191

courseID	dept	number	student ID
06902	CSC	221	1123
06902	CSC	221	1129
06902	CSC	221	1145
06903	CSC	231	1123
06903	CSC	231	1145
06904	CSC	241	1123
06904	CSC	241	1129

Distinguish from: SELECT * FROM Course c, Enrollment e where c.courseID = e.courseID which would preserve both courseID columns

SQL: Semi Joins

No specific SEMIJOIN statement –why?

- Use SELECT clause to manage this…
- SELECT DISTINCT s.* FROM Student s, Enrollment e WHERE s.studentID = e.studentID

  Shows all student information for any students enrolled in any course

SQL: Insert

- How to add data?
- Initially: Add by specifying values associated with attributes:

INSERT INTO tableName [(columnList,…)] VALUES(value,…);

- String-like values should be single quoted
- Numerical values should not be quoted
- Can use NULL

INSERT INTO enrollment(courseID, studentID) VALUES (06902,1123);

Example Hotel Queries

Provide details of all single rooms that are priced under 100.00:

SELECT * FROM room WHERE price < 100.00 AND type='S'

Provide names and addresses of all guests:

SELECT guestName,guestAddress FROM guest;

List the price and type of all rooms at the Winston hotel:

We argued all three of these should work. The first is the least computationally efficient, since it requires a large Cartesian product. The last is probably the most efficienct, since it never generates a Cartesian product.

SELECT price,type FROM room r, hotel h WHERE r.hotelNumber = h.hotelNumber AND h.hotelName = 'Winston';

SELECT price,type FROM room NATURAL JOIN (SELECT * FROM hotel WHERE hotelName='Winston') as WH;

SELECT price,type FROM room WHERE hotelNumber=(SELECT hotelNumber FROM hotel WHERE hotelName = 'Winston') as WH;