SQL

CSCI435

Project

- Due 4pm, May 19
- Send your report through email
  - Functional requirement (problem definition)
  - ER diagram
  - SQL for the creation of each table
  - SQL output of the description of each table
  - SQL output of three rows for each table
- Grading: Total 40
  - Design (ER diagram): 20 points
  - Implementation: 15 points
  - Difficulty: 5 points

Joins in SQL

- NATURAL JOIN
- Regular θ JOIN
- LEFT OUTER JOIN
- RIGHT OUTER JOIN
- CROSS JOIN

INNER JOIN syntax

```
SELECT [DISTINCT | ALL]
{* | [column-expression [AS new-name]] [,…]
    | [table-name.column-name [AS new-name]] [,…]
    | [table-id.column-name [AS new-name] [,…]}
FROM [table-name1 [[AS] table-id1] [,…]]
[INNER JOIN table-name2 [[AS] table-id2]]
{ON {column-name1
      | table-name1.column-name1
      |  tableid1.column-name1}
   = {column-name2
      | table-name2.column-name2
      | table-id2.column-name2}
| USING (column-name […,])}]
```

Example

use bank;

SELECT * FROM employee e **INNER JOIN** department d **ON e.**dept_id=d.dept_id;

- You have to provide unambiguous column names

SELECT * FROM employee **INNER JOIN** department **USING** (dept_id);

Self joins

- You can join a table to itself

SELECT e.fname, e.lname, e_mgr.fname mgr_fname, e_mgr.lname mgr_lname

FROM employee e **INNER JOIN** employee e_mgr

ON e.superior_emp_id = e_mgr.emp_id;

Reusing a table in joins

- If you need a table for more than one join, you have to give it more than one name
- What does this do?

```
SELECT a.account_id, e.emp_id, b_a.name open_branch,
    b_e.name emp_branch
FROM account a
INNER JOIN branch b_a ON a.open_branch_id = b_a.branch_id
INNER JOIN employee e ON a.open_emp_id = e.emp_id
INNER JOIN branch b_e ON e.assigned_branch_id =
b_e.branch_id
WHERE a.product_cd = 'CHK';
```

Non-equijoins

- The syntax

  … FROM [*table-name1* [[AS] *table-id1*]

  INNER JOIN *table-name2* [[AS] *table-id2*]]

  ON … = …

assumes that an equality is being tested

- It is possible, however, to join tables without a foreign key relationship, using a comparison operator instead
- In that case the syntax is

… FROM [*table-name1* [[AS] *table-id1*]

INNER JOIN *table-name2* [[AS] *table-id2*]]

ON … *comparison-operator* …

Syntax for other JOINs

table_reference [INNER|CROSS] JOIN *table_reference* [*join_condition*]
| *table_reference* LEFT [OUTER] JOIN *table_reference* *join_condition*
| *table_reference* RIGHT [OUTER] JOIN *table_reference* *join_condition*
| *table_reference* NATURAL JOIN *table_reference*

where *table_reference* is

tbl_name [[AS] *alias*]

where *join_condition* is

ON *conditional_expr* | USING (*column_list*)

OUTER JOIN

- INNER JOIN produces one row for each pair of matching rows, but NULLs *never* match
- OUTER JOIN combines ALL records form one table with either matching records from the second table (if any) or with NULL (if not).
  - LEFT JOIN – ALL records from table to left of word "JOIN"
  - RIGHT JOIN – ALL records from table to right of word "JOIN"

OUTER JOINs

- *table-1* LEFT OUTER JOIN *table-2* generates a row for every match

  from the left table, even if table-2 has no match

```
SELECT c.cust_id, i.fname, i.lname
FROM customer c LEFT OUTER JOIN individual i
ON c.cust_id = i.cust_id;
```

- *table-1* RIGHT OUTER JOIN *table-2* generates a row for every match from the right table, even if table-1 has no match

```
SELECT c.cust_id, i.fname, i.lname
FROM customer c RIGHT OUTER JOIN individual i
ON c.cust_id = i.cust_id;
```

NULLs in JOINs

- Often data is missing or fails to match
- Example: try joining the bank's individual customer table to its account table (how many accounts and customers?)

```
SELECT a.account_id, c.cust_id FROM account a
INNER JOIN individual c
 ON a.cust_id = c.cust_id;
```

- If you want to process NULL values you need to use an OUTER JOIN

Example

- What are the customer ID, federal ID, customer type, and name of all customers (individual and business)?

```
SELECT c.cust_id, c.fed_id, c.cust_type_cd,
  concat(i.fname,' ',i.lname) person_name,
  b.name business_name
FROM customer c
LEFT OUTER JOIN individual i ON c.cust_id =
  i.cust_id
LEFT OUTER JOIN business b ON c.cust_id =
  b.cust_id;
```

Substring comparison

- In SQL, character string attribute values are ***not*** really atomic
- **LIKE** comparison operator compares partial strings
- Syntax: LIKE *'%expression%*'
- Two reserved characters are used: '**%**' (or '*****' in some implementations) replaces an arbitrary number of characters, and '**_**‘ replaces a single arbitrary character
- Example: List all employees who live in Houston, Texas

  SELECT FNAME, LNAME

  FROM EMPLOYEE

  WHERE ADDRESS **LIKE** '%Houston,TX%';

Set operations

- Include **UNION** (for ∪), **MINUS** (for set difference) and **INTERSECT** (for ∩)
- Results from these set operations are ***sets*** of tuples, i.e., ***duplicate tuples are eliminated***
- Apply only to ***union compatible relations*** (same attributes in the same order)
- Example: Make a list of all project numbers for projects that involve an employee whose last name is 'Smith' as a worker or as a manager of the department that controls the project.

```
SELECT PNAME
FROM PROJECT,DEPARTMENT,EMPLOYEE
WHERE DNUM=DNUMBER AND MGRSSN=SSN AND LNAME='Smith'
UNION
SELECT PNAME
FROM PROJECT, WORKS_ON, EMPLOYEE
WHERE PNUMBER=PNO AND ESSN=SSN AND NAME='Smith';
```

Set operator precedence

- INTERSECT has precedence over UNION and EXCEPT
  - $A \cap B \cup C = (A \cap B) \cup C$
  - Example: A = {1,2,3}, B = {2,3,4}, C= {2,4,6}

Set operator precedence

- INTERSECT has precedence over UNION and EXCEPT
  - $A \cap B \cup C = (A \cap B) \cup C$
  - Example: A = {1,2,3}, B = {2,3,4}, C= {2,4,6}
  - $A \cap (B \cup C) = ?$

Set operator precedence

- INTERSECT has precedence over UNION and EXCEPT
  - $A \cap B \cup C = (A \cap B) \cup C$
  - Example: A = {1,2,3}, B = {2,3,4}, C= {2,4,6}
  - $A \cap (B \cup C) = A \cap \{2,3,4,6\} = \{2,3\}$

Set operator precedence

- INTERSECT has precedence over UNION and EXCEPT
  - $A \cap B \cup C = (A \cap B) \cup C$
  - Example: A = {1,2,3}, B = {2,3,4}, C= {2,4,6}
  - $A \cap (B \cup C) = A \cap \{2,3,4,6\} = \{2,3\}$
  - $(A \cap B) \cup C$= ?

Set operator precedence

- INTERSECT has precedence over UNION and EXCEPT
  - $A \cap B \cup C = (A \cap B) \cup C$
  - Example: A = {1,2,3}, B = {2,3,4}, C= {2,4,6}
  - $A \cap (B \cup C) = A \cap \{2,3,4,6\} = \{2,3\}$
  - $(A \cap B) \cup C = \{2,3\} \cup C = \{2,3,4,6\}$
- Multiple parentheses recommended for clarity (where supported)

MINUS and INTERSECT are not supported in MySQL 5.1

Subqueries

- ***Subquery*** (or ***nested query***) is a query contained within another SQL statement
- Subquery evaluates to a table
- Always parenthesized
- Can return a constant
- Number of rows and columns in resultant table determines how
- subquery can be used
- To determine this, ***always test subquery alone first***
- After the containing statement is executed, the table produced by the subquery is discarded
- Can have several levels of nested queries**!**

Subquery syntax

- Syntax (select-statement [**AS** *some-name*])
- Example: What data is associated with the largest account_id?

```
SELECT * FROM account WHERE account_id =
(SELECT MAX(account_id) FROM account);
```

- Many of previous queries can be specified with nesting instead
- Example: Retrieve the name and address of all employees who work for the 'Research' department

```
SELECT FNAME, LNAME, ADDRESS
FROM EMPLOYEE
WHERE DNO IN
      (SELECT DNUMBER
      FROM DEPARTMENT
      WHERE DNAME='Research');
```

- Here the nested query is *not correlated* with the outer query**!**

Subqueries as tables

- It is possible to use an intermediate table, created by a subquery, in a join
- Give the subquery an id

```
SELECT a.account_id, a.cust_id, a.open_date, a.product_cd
FROM account a INNER JOIN
(SELECT emp_id, assigned_branch_id
FROM employee
WHERE start_date <= '2003-01-01'
AND (title = 'Teller' OR title = 'Head Teller')) e
ON a.open_emp_id = e.emp_id
INNER JOIN
(SELECT branch_id FROM branch
WHERE name = 'Woburn Branch') b
ON e.assigned_branch_id = b.branch_id;
```

Practice

- **1. Which individual customers (names) have an available balance in some account that is different from the pending balance there?**
- **2. Which employee(s) open checking accounts but not savings accounts?**
- **3. Which branches have had customers open Customer Accounts?**
- **4. Display, from a single query, the number of countries that speak 1 official language, 2 official languages, and so on. (Hint: use world database)**

Practice

- **1. Which individual customers (names) have an available balance in some account that is different from the pending balance there?**

select i.fname, i.lname, a.avail_balance, a.pending_balance from account a inner join individual i on a.cust_id=i.cust_id where a.avail_balance != a.pending_balance;

- **2. Which employee(s) open checking accounts but not savings accounts?**

select e.fname, e.lname from employee e inner join (select open_emp_id from account
where product_cd = 'CHK' and open_emp_id not in (select open_emp_id from account
where product_cd = 'SAV' and *open_emp_id is not null*)) s on e.emp_id = s.open_emp_id;

- **3. Which branches have had customers open Customer Accounts?**

select distinct b.name from branch b inner join account a on a.open_branch_id = b.branch_id inner join product p on a.product_cd = p.product_cd inner join product_type pt on p.product_type_cd = pt.product_type_cd where pt.name like '%customer accounts%';

- **4. Display, from a single query, the number of countries that speak 1 official language, 2 official languages, and so on. (Hint: use world database)**

select ol_grp.num_lan, count(ol_grp.cc) from
(select CountryCode cc, count(ol.Language) num_lan from CountryLanguage where IsOfficial = 'T' group by ol.CountryCode) ol_grp group by ol_grp.num_lan;