CSCI 403
Database Management

12 – Miscellaneous Topics

Topics

This lecture is for stuff I forgot or didn't have time to cover so far…

- Miscellaneous SELECT
  - DISTINCT
  - JOIN clause and outer joins
  - SET operations
  - WITH (Common Table Expression) queries
  - Other stuff
- Bulk loading and COPY
- Workflow

DISTINCT

DISTINCT

- SELECT queries may return duplicate rows
  - E.g.

    SELECT instructor FROM mines_courses;
- DISTINCT keyword lets us remove duplicates:

  SELECT DISTINCT instructor FROM mines_courses;
- Also, can use with aggregates (not all DBMSes):

  SELECT COUNT (DISTINCT instructor) FROM mines_courses;

  SELECT AVG(DISTINCT enrollment) FROM mines_courses;

  (the last is a silly application, just for illustration)

JOINS

JOIN Clause

We've so far been using "WHERE clause" joins:

```
SELECT a.x, b.y from a, b WHERE a.z = b.z;
```

We can also use a JOIN expression explicitly:

```
SELECT a.x, b.y
FROM a INNER JOIN b ON (a.z = b.z);
```

More JOIN

Can use with multiple conditions:

```
SELECT a.x, b.y
FROM a INNER JOIN b ON (a.z = b.z AND a.zz = b.zz);
```

Can use with multiple tables:

```
SELECT a.x, b.y, c.z
FROM a INNER JOIN b ON (a.foo = b.foo)
        INNER JOIN c ON (b.bar = c.bar);
```


INNER JOIN Notes

- The INNER keyword is optional (INNER is the default)
- Performance is same using WHERE clause or INNER JOIN
- Choice between WHERE clause or JOIN clause is completely up to you:
  - Personal preference
  - Readability (JOIN clause can get complicated with many tables)

OUTER JOIN

- Inner joins only return rows that match join condition:
  - What if we want *all* rows from one or both tables?
  - Outer join lets us get everything, pairing up rows where possible.
  - Example (in class):

```
SELECT mc.course_id, mc.section, mc.title,
mc.instructor, mcf.email, mcf.office
FROM mines_courses AS mc
     LEFT OUTER JOIN mines_cs_faculty AS mcf
     ON (mc.instructor = mcf.name);
```


LEFT and RIGHT

```
SELECT mc.course_id, mc.section, mc.title,
mc.instructor, mcf.email, mcf.office
FROM mines_courses AS mc
     LEFT OUTER JOIN mines_cs_faculty AS mcf
     ON (mc.instructor = mcf.name);
```

The above gives all rows from mines_courses (the left-hand table in the join).

RIGHT does the opposite.

Note that OUTER is optional – LEFT and RIGHT imply OUTER.

FULL OUTER JOIN

As the name implies, gives all rows from *both* tables in join, matching rows where possible.

Same example as above, replacing LEFT with FULL (OUTER is optional again).

To see effects, first have to insert a bogus faculty member into mines_cs_faculty…

Applications of Outer Joins

Great for finding missing data, e.g., data where a foreign key is null, by using IS NULL in WHERE:

```
SELECT mc.course_id, mc.section, mc.title,
mc.instructor, mcf.email, mcf.office
FROM mines_courses AS mc
     LEFT OUTER JOIN mines_cs_faculty AS mcf
     ON (mc.instructor = mcf.name)
WHERE mcf.name IS NULL;
```

Oracle Outer Joins

Oracle defines a special operator that allows left and right outer joins to be created using WHERE clause (very handy!):

```
SELECT a.x, b.y
FROM a, b
WHERE a.z = b.z(+);   -- right outer join!
```


SET OPERATIONS

Union, Intersection, Difference

You can perform set operations on two or more SELECT query results:

```
SELECT course_id, title, instructor
FROM mines_courses
UNION
SELECT 'CSCI999', 'CS Fun Fair', name
FROM mines_cs_faculty;
```

Column names come from first SELECT query.
Column types & count have to match.

Set Operators

Set union : UNION

Set intersection : INTERSECT

Set difference: EXCEPT

[ALL]

- By default, set operators imply DISTINCT
  - This is because sets, mathematically, contain only distinct units
- To avoid this behavior, add ALL keyword

```
SELECT …
UNION ALL
SELECT …
```


WITH

WITH (Common Table Expression) Queries

- An alternative to subqueries, also with some cool applications
- Effectively, provides a temporary named relation for use in a query
- Example:

```
WITH cs_courses AS (
     SELECT * FROM mines_courses
     WHERE course_id LIKE 'CSCI%'
)
SELECT DISTINCT course_id, title
FROM cs_courses
WHERE instructor = 'Paone, Jeffrey';
```


WITH (PostgreSQL extensions)

- WITH in PostgreSQL has some powerful extensions:
  - Can use INSERT/UPDATE/DELETE in CTE
    - Gives an alternative to doing transactions (a later topic)
    - RETURNING clause (PostgreSQL only) also useful here
  - Can use with INSERT/UPDATE/DELETE
- Example:

```
WITH q AS (
  DELETE FROM products WHERE fruit = 'apple'
)
INSERT INTO products
VALUES ('apple', 'FruitCo', 3.59);
```


RETURNING clause (PostgreSQL only)

Since I mentioned it in previous slide:

RETURNING returns modified rows from an INSERT, UPDATE, or DELETE:

```
DELETE FROM products
WHERE fruit = 'orange'
RETURNING *;
```

Very useful for capturing, e.g., result of serial column default values after INSERT!

Recursive WITH

Recursion in SQL!

General form:

```
WITH RECURSIVE name AS (
    <non-recursive SELECT query>
    UNION
    <recursive SELECT query (references name)>
)
SELECT …
```


Recursive Example

From our database (table from textbook):

```
WITH RECURSIVE emp_rec (padding, first, last, ssn) AS (
    SELECT '', fname, lname, superssn
    FROM employee
    WHERE fname = 'Joyce' AND lname = 'English'
  UNION ALL
    SELECT er.padding || '  ', e.fname, e.lname, e.superssn
    FROM emp_rec er, employee e
    WHERE e.ssn = er.ssn
)
SELECT padding || first || ' ' ||  last FROM emp_rec;
```


OTHER STUFF

OFFSET…FETCH…

Get some rows starting at some offset:

```
SELECT * FROM mines_cs_faculty
ORDER BY name
FETCH FIRST 5 ROWS ONLY;

SELECT * FROM mines_cs_faculty
ORDER BY name
OFFSET 5
FETCH FIRST 5 ROWS ONLY;
```


BULK LOADING

Bulk Loading

- Term for loading lots of data into database efficiently
- Compare to doing millions of INSERT statements…
- Typical workflow (sometimes known as ETL: Extract, Transform, Load):
  - Acquire data in text, .csv, JSON, or other format
  - Preprocess if needed/desired using non-SQL tools
  - Bulk load data as is (dirty, incomplete, incorrectly formatted) into staging table(s)
  - Postprocess if needed (clean up, format) using SQL
  - Use SQL to load into final table(s)

Bulk Loading in PostgreSQL: COPY

- COPY command in PostgreSQL SQL
  - Fast bulk loading from various formats
  - Lets *DB administrators only* load data from server filesystem
- \COPY in psql
  - Fast bulk loading across network
  - Lets non-administrators bulk load from local filesystem
- Other bulk loaders/ETL tools
  - Commercial and free tools available
  - Google "ETL postgresql"

Lessons learned the hard way

WORKFLOW

Dangers of Command Line SQL

- Databases are dynamic storage
  - You can modify data quickly with SQL
  - Everyone sees modified data immediately
- SQL is immediate, irrevocable
  - What if you make a mistake?
  - E.g., production database migration (new schema)
- Don't do command line SQL except for exploratory programming on dev DB!

Scripts Are Your Friends

- Keep dev, test, production databases (at minimum)
- During development, write & test scripts on dev
- When time for a software release/database migration:
  - Clone a new db from test
  - Apply scripts
  - Fix scripts
  - Repeat as necessary
- Scripts go in your version control system (git)!!
  - In theory, should be able to reproduce prod database schema (not data, necessarily) from scratch by running all scripts from beginning of project.

DBAs/SysAdmins Are Also Friends

- Your production database should be backed up *nightly*
  - Not for crashes, necessarily, as SQL databases can recover from those on their own
  - Instead, need it to recover from developer mistakes ☺
- However, make sure DBAs/sysadmins *test* backups regularly (DB backup is tricky!)

Up Next

- Next lecture:
  - Database modeling and design
  - Entity-Relationship Diagrams (ERD)
  - Reading: Chapter 3, "Data Modeling Using the Entity-Relationship Model"