Course Summary

CSCC43 - Introduction to Databases

Fall 2019

Computer & Mathematical Sciences
UNIVERSITY OF TORONTO
SCARBOROUGH

Instructor:	**Sina Meraji**
Email:	camelia.karimian@utoronto.ca
Office:	TBA
Office Hours:	Thr 17:00 - 18:00

1 Relational Algebra

1.1 Notations

Select: $\sigma_{condition}(R)$
where condition is a Boolean expression, the select operator results in a new relation.

Project: $\pi_{attributes}(R)$
where attributes are columns of a table, the project operator results in a new relation.

Cartesian Product: $R_1 \times R_2$
The Cartesian product operator results in a relation with every combination of tuples from R_1 concatenated with a tuple from R_2

Natural Join: $R_1 \bowtie R_2$
The natural join operator joins two relations by matching a column that exists in both (determined by name).
Natural join is both commutative and associative.

Theta Join: $R_1 \bowtie_{condition} R_2$
Theta join is equivalent to $\sigma_{condition}(R_1 \times R_2)$

Assignment Operator: $R_1(A_1, A_2, \ldots, A_n) :=$ Some Expression
Where $A_1, \ldots, A_n$ are attributes of the new relationship.

Rename Operator: $\rho_{R_1}(R_2)$
The above renames the relationship R_2 to R_1, ρ can also be used to rename attributes on the fly.
such as $\rho_{R_1(A_1,\ldots,A_n)}(R_2)$, useful when renaming within an expression.

2 Foreign Keys

Let R_1, R_2 be relations with attributes $X \in R_1$ and $Y \in R_2$
We can declare a foreign key constraint like: $R_1[X] \subseteq R_2[Y]$
if Y is a key in R_2 and X, Y have the same arity, then X is a foreign key in R_1.

3 Queries

General types of queries:

1. **Maximum/Minimum**

   Cartesian Product the tuples, find those that are **not** the max. Then subtract from all to find the max(s).

2. **k or more**

   Make all combinations of k different tuples that satisfy the condition.

3. **exactly k**

   (k or more) - (k + 1) or more

4. every

   Subtract failures from all to get the answer.

4 SQL

4.1 Data Definition Language

DDL is used for defining schemas.
Creating a table:

```
CREATE TABLE table_name (
    column_name TYPE column_constraint,
    table_constraint
) INHERITS existing_table_name;
```

Inherit is used when you want this new table to contain all columns of an existing table.

Column Constraints

- **NOT NULL**, The value of this column cannot be NULL.
- **UNIQUE**, The value of the column must be unique across the whole table. However, the column can have many **NULL** values.
- **PRIMARY KEY**, This constraint is simply the combination of **NOT NULL** and **UNIQUE**. To declare a Primary Key across multiple columns, you must use the table_constraint
- **CHECK**, This enables a check condition when you insert data, for example the GPA column might have a check(value >= 0 and value <= 4.0).
- **REFERENCES**, Constraints the value of the column that exists in a column in another table. This is used to define foreign keys.

Table Constraints

- **UNIQUE (column_list)**, enforces unique values in the columns listed inside parentheses.
- **PRIMARY KEY(column_list)**, to define the primary key that consists of multiple columns.
- **CHECK (condition)** check a condition when inserting or updating data.
- **REFERENCES**, restraining value stored in the column that must exist in a column in another table.

4.2 Data Manipulation Language

DDL is used for writing queries and modifying the database.

4.3 SELECT

The select statement is used to retrieve any data from a table.
To select all rows of a table, the '*' is used as shorthand for "all columns"
To specify columns, see line 2.

```
0 SELECT * FROM table;
1 SELECT column_1, column_2 FROM table;
2 SELECT city, (temp_lo + temp_hi)/2 AS temp_avg, date FROM weather;
```

You can also write expressions, not just column references
The **AS** clause is used to relabel the output column.

4.4 WHERE

A query can be ”qualified” by adding a WHERE clause that specifies which rows are wanted. The WHERE clause should contain a Boolean expression, then only rows for which Boolean expression evaluates true are returned.
The expression can contain the usual Boolean Operators (**AND**, **OR** and **NOT**).

```
SELECT date as rainy_days_in_SF FROM weather
   WHERE prcp > 0.0 AND city = ’San Francisco’;
```

4.5 ORDER BY

The order by statement is used to determine the output order from a select statement. For example:

```
SELECT * FROM weather
   ORDER BY city, temp_lo DESC;
```

The output will first sort the output by city (alphabetically, ascending), then for the same city it sorts by temp_lo(numerically, descending).
To specify order by in descending order, attach **DESC** after the column.
You can also order by an expression, for example **ORDER BY prcp+temp_lo DESC**.

4.6 LIKE

Used for string comparisons, comparing values of a column to a **Pattern**.
Patter: A quoted string, where %: denotes any matching string, and _: denotes any single character.

```
SELECT *
FROM Course
WHERE name NOT LIKE "%Mat%";
```

5 AGGREGATION

Aggregation is used when we wish to compute something across the values in a column.
Functions such as **SUM, AVG, COUNT, MIN** and **MAX** can be applied to a column in a **SELECT** clause.
In order to stop duplicates from contributing to the aggregation, use keyword **DISTINCT** inside the aggregation brackets. Note that aggregate ignores nulls.
Examples of Aggregate:

```
SELECT avg(grade) FROM took;
SELECT min(grade), max(grade), sum(grade), avg(grade)
FROM runnymede WHERE name > ’cate’;
```

Remember **SELECT** is always the last thing done, so in the second example above, the aggregate happens on rows where the name is after cate (alphabetically).

5.1 GROUP BY

```
SELECT oid, avg(grade) FROM took
GROUP BY oid;
```

HAVING, the having clause lets you decide which groups to keep, recall the **WHERE** clause allows you to decide which tuples.
The **HAVING** clause may only refer to attributes if they are either aggregated or being grouped by.

```
SELECT oid, avg(grade) FROM took
GROUP BY oid HAVING avg(grade) < 80;
```

5.2 SET OPERATIONS

the typical **SELECT-FROM-WHERE** statements leave duplicates in unless you specify **DISTINCT**.
The set operation clauses are **UNION, INTERSECTION** and **EXCEPT**. Where except is set subtraction.
Individual sub-queries must be expressed within **brackets**.

```
(SELECT sid FROM Took WHERE grade > 90)
UNION
(SELECT sid FROM Took WHERE grade < 50);
```

Note that set operations require sub-queries to have the same arity. Duplicates will always be eliminated from the result.
In order to keep duplicates we can force by using **UNION ALL**.

6 VIEWS

Views can be used to break down a larger query. The type of view PostgreSQL uses is called **Virtual View**. Where no tuples are stored, and its simply a defined query for constructing a relation when needed.

```
CREATE VIEW view_name AS
[SELECT-FROM-WHERE];

SELECT * FROM view_name;
```

To delete the view, simply use the **DROP** clause.

7 Schemas

Schema: is a kind of namespace, where everything defined (tables, types, etc) goes into one big pot.
Useful for logical organization and avoiding name clashes.
By default, PSQL has a schema called ”public”.
You can also create your own for. Example:

```
CREATE SCHEMA University;
```

Then to refer things inside a particular schema, you can use the dot notation:

```
CREATE TABLE University.Student (...);
SELECT * FROM University.Students;
```

If you refer to a name without specifying what schema, then it goes into the schema called "public", this is saying that when creating a table called **"frindle"** you are actually defining **public.frindle**.

To remove a schema use:

```
DROP SCHEMA University CASCADE;
```

The CASCADE keyword means everything inside it is dropped too. to avoid error messages use the **"if exists"**.

Usage Pattern

```
DROP SCHEMA IF EXISTS University CASCADE;

CREATE SCHEMA University;

SET SEARCH_PATH TO University;
```

Helpful during development, when you may want to change the schema or test queries under different conditions.

The typical **workflow** is as follows:

- Create DDL file with the schema.
- Create a file that inserts contents into the database.
- Import the files onto the postgreSQL shell.
- Run quries directly in the shell or by importing queries written in files.

8 DDL

When creating a table, you must define the type of each attribute.

8.1 built-in types

- **CHAR(n)**: Fixed length string of n characters, padded with blanks when necessary.
- **VARCHAR(n)**: Variable length string of up to n characters.
- **TEXT:** variable-length, unlimited. Not in SQL standard but supported by PSQL.
- **INT:** integer
- **FLOAT:** real number
- **DATE; TIME; TIMESTAMP** (date plus time)

8.2 Domain

User-defined types: it is possible to make a more specific version of a type by defining constraints and perhaps a default value.
Example:

```
CREATE DOMAIN Grade AS INT
    DEFAULT null
    check (value >= 0 and value <= 100);

CREATE DOMAIN Campus AS VARCHAR(4)
    default 'StG'
    check (value in ('StG', 'UTM', 'UTSC));
```

constraints on a type are checked every time a value is assigned to an attribute of that type.

The default value of a type is used when no value has been specified. This is useful as we can run a query and insert the resulting tuples into a relation without giving values for all attributes.
Furthermore, table attributes can also have default values.
The difference:

- Attribute Default is for attributes in that one table
- Type Default is for every attribute defined of that type

8.3 Keys/Foreign Keys

Declaring a set of one or more attributes are the **PRIMARY KEY** for a relation means:

- They form a unique key
- Their values will never be null (this does not need to be separately declared).

Each column must have 0 to 1 primary key. You cannot have more than one primary key but its possible to have no primary keys.
For a single attribute key, can be part of the attribute definition./