Computing for Medicine (C4M)
Seminar 3: Databases

Michelle Craig
Associate Professor, Teaching Stream
mcraig@cs.toronto.edu

UNIVERSITY OF TORONTO

Relational Model

- The relational model is based on the concept of a relation or table.
- Some example relations:

Students

SID	Surname	Campus
99999	Horton	StG
54287	Daniels	UTM
68000	Smith	StG

Offerings

OID	CID	term	Instructor
8	4	20179	Craig
9	4	20169	Campbell
10	3	20179	Law

Took

SID	OID	Grade
99999	8	90
54287	8	50
68000	10	88
99999	10	87
99999	5	95
68000	3	90
87654	4	80

SQL: Structured Query Language

- SQL is a very high-level language.
  - Say "what" rather than "how."
- You write queries without manipulating data.
  - Different than Python
- Provides physical "data independence:"
  - Details of how the data is stored can change with no impact on your queries

PostgreSQL

- We'll be working in PostgreSQL, an open-source relational DBMS.
- Documentation is good and very helpful.
- Download free version or use on teach.cs account

Basic query with one relation

```
SELECT gender
FROM Patients
WHERE subject_id = 45;
```

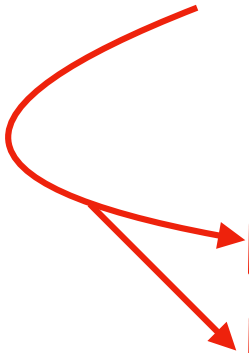
```
SELECT subject_id, hadm_id
FROM Admissions
WHERE admission_type = 'URGENT';
```

subject_id	hadm_id	admission_type	…
		‘URGENT’	
		URGENT’	

Meaning of a query with one relation

```
SELECT gender
FROM Patients
WHERE subject_id = 45;
```

```
SELECT subject_id, hadm_id
FROM Admissions
WHERE admission_type = 'URGENT';
```

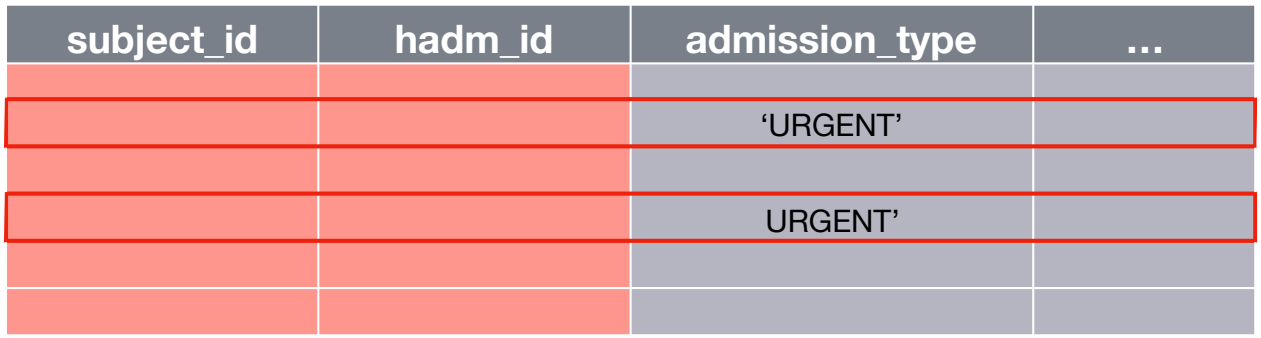


subject_id	hadm_id	admission_type	...
		‘URGENT’	
		URGENT’	

Meaning of a query with one relation

```
SELECT gender
FROM Patients
WHERE subject_id = 45;
```

```
SELECT subject_id, hadm_id
FROM Admissions
WHERE admission_type = 'URGENT';
```



Meaning of a query with one relation

```
SELECT gender
FROM Patients
WHERE subject_id = 45;
```

```
SELECT subject_id, hadm_id
FROM Admissions
WHERE admission_type = 'URGENT';
```

subject_id	hadm_id

… and with multiple relations

```
SELECT gender
FROM Patients p, Admissions a
WHERE p.subject_id = a.subject_id
AND dob = '1990-10-16 00:00:00';
```

… and with multiple relations

```
SELECT gender
FROM Patients p, Admissions a
WHERE p.subject_id = a.subject_id
AND dob = '1990-10-16 00:00:00';
```

- Every record in Patients is joined with every record in Admissions
- Even if it doesn't make sense to put them together
- Then the condition in the WHERE clause removes the non-sense rows
- Optional renaming is convenient

* In SELECT clauses

- A * in the SELECT clause means "all attributes of this relation."
- Example:

```
SELECT *
FROM ICUStays
WHERE EXTRACT (month FROM intime) = 1;
```

Complex Conditions in a WHERE

- We can build boolean expressions with operators that produce boolean results.
  - comparison operators: `=, <>, <, >, <=, >=`
  - and many other operators
- Note that "not equals" is unusual: `<>`
- We can combine boolean expressions with:
  - Boolean operators: `AND, OR, NOT`.

ORDER BY

- To put the tuples in order, add this as the final clause:

  `ORDER BY «attribute list» [DESC]`
- The default is ascending order; DESC overrides it to force descending order.
- The ordering is the last thing done before the SELECT, so all attributes are still available.

Expressions in SELECT clauses

- Instead of a simple attribute name, you can use an expression in a SELECT clause.
- Can rename the resulting column with AS
- Examples:

```
SELECT sid, grade+10 AS adjusted
FROM Took;
```

Pattern operators

- Two ways to compare a string to a pattern by:
  - *«attribute»* `LIKE` *«pattern»*
  - *«attribute»* `NOT LIKE` *«pattern»*
- Pattern is a quoted string
  - `%` means: any string
  - `_` means: any single character
    - Example:

```
SELECT *
FROM Offerings
WHERE term LIKE '2017%';
```

Computing on a column

- We often want to compute something across the values in a column.
- `SUM`, `AVG`, `COUNT`, `MIN`, and `MAX` can be applied to a column in a SELECT clause.
- Also, `COUNT(*)` counts the number of tuples.
- We call this aggregation.
- Note: To stop duplicates from contributing to the aggregation, use `DISTINCT` inside the brackets. (Does not affect `MIN` or `MAX`.)
- Example: aggregation.txt

Grouping

- Example: group-by.txt
- If we follow a SELECT-FROM-WHERE expression with GROUP BY <attributes>
  - The tuples are grouped according to the values of those attributes, and
  - any aggregation gives us a single value per group.

Restrictions on aggregation

- If any aggregation is used, then each element of the SELECT list must be either:
  - aggregated, or
  - an attribute on the GROUP BY list.
- Otherwise, it doesn't even make sense to include the attribute.

HAVING Clauses

- Example: having.txt
- WHERE let's you decide which tuples to keep.
- Similarly, you can decide which *groups* to keep.
- Syntax:

```
...
GROUP BY «attributes»
HAVING «condition»
```

- Semantics:
  Only groups satisfying the condition are kept.

Tables can have duplicates in SQL

- A table can have duplicate tuples, unless this would violate an integrity constraint.
- And SELECT-FROM-WHERE statements leave duplicates in unless you say not to.
- Why?
  - Getting rid of duplicates is expensive!
  - We may want the duplicates because they tell us how many times something occurred.

Union, Intersection, and Difference

- These are expressed as:

  ```
  («subquery») UNION («subquery»)
  («subquery») INTERSECT («subquery»)
  («subquery») EXCEPT («subquery»)
  ```

- The brackets are mandatory.
- The operands must be queries; you can't simply use a relation name.
- By default these operations use SETS so duplicates are removed.

Example

```
(SELECT sid
 FROM Took
 WHERE grade > 95)
          UNION
(SELECT sid
 FROM Took
 WHERE grade < 50);
```

Motivation: Efficiency

- When doing WHERE it is easier not to eliminate duplicates.
  - Just work one tuple at a time.
- For intersection or difference, it is most efficient to sort the relations first.
  - At that point you may as well eliminate the duplicates anyway.
- We can force the result of a SFW query to be a set by using `SELECT DISTINCT ...`

Views

- A view is a relation defined in terms of stored tables (called base tables) and other views.
- Access a view like any base table

- Use it to break down a large query

Example: defining a view

- A view for students who earned an 80 or higher in a CSC course.

```
CREATE VIEW topresults AS
SELECT firstname, surname, cnum
FROM Student, Took, Offering
WHERE
    Student.sid = Took.sid AND
    Took.oid = Offering.oid AND
    grade >= 80 AND dept = 'CSC';
```

Database Modifications

- Queries return a relation.
- A modification command does not; it changes the database in some way.
- Three kinds of modifications:
  - Insert a tuple or tuples.
  - Delete a tuple or tuples.
  - Update the value(s) of an existing tuple or tuples.

Insert tuples

- `INSERT INTO` *«table»* `VALUES` *«list of rows»*`;`

```
INSERT INTO Patients VALUES (2000, 2000,
'F', '2017-01-01', Null);
```

Updates

- To change the value of certain attributes in certain tuples to given values:

```
UPDATE «relation»
SET «list of attribute assignments»
WHERE «condition on tuples»;
```

Example: update one tuple

- Updating one tuple:

```
UPDATE Student
SET campus = 'UTM'
WHERE sid = 99999;
```

- Updating several tuples:

```
UPDATE Took
SET grade = 50
WHERE grade >= 47 and grade < 50;
```