

Recap

- Data-savviness is the future!
- Notion of a DBMS
- The relational data model and algebra: bags and sets
- **Next: SQL**
 - Will be easier now that you understand relational algebra



SQL: Structured Query Language (or “sequel”)

- High level “declarative” language supported by relational databases
 - You avoid specifying many of the data manipulation details that would be necessary in a language like Python or C++
- The user issues “SQL queries”
- But it is restricted
 - Can’t do everything you can do in Python (but you can do a lot!)
 - This restriction is by design: allows the system to automatically optimize the queries
- It is still quite powerful: can do most things you want to do with data



The Basic Form: Select From Where

- Basic Form:

SELECT attributes

FROM tables

WHERE condition about tuples in tables

If you're trying to map this to relational algebra, beware that SELECT is projection, and WHERE is selection! This confusion has sadly persisted across decades...



DVDRental Dataset

- Many tables: actor, address, country, film, inventory, payment, staff...
- DEMO!
- `SELECT * FROM Actor:` `*` is a shorthand for “all”
- `SELECT * FROM Film`
- `SELECT title FROM Film:`
 - only one attribute selected: title
- `SELECT * FROM Film WHERE release_year=2006 AND length <100;`
 - two conditions met by tuples



How to Read a SQL Query

- Start with the FROM clause: lists the tables
 - (so far only one, we'll get to more soon)
- Apply the selection in the WHERE clause
- Apply the projection in the SELECT clause
- Convention: keywords caps, attribs small, rel first name caps



DVDRental Dataset

- `SELECT title AS film_name, release_year, length/60 FROM Film`
 - What can you do with a `SELECT` clause:
 - Renaming attributes (`AS`)
 - Expressions
- `SELECT * FROM Film WHERE release_year=2006 AND length <100;`
- `SELECT title FROM Film WHERE title LIKE 'The%';`
 - What can you do with a `WHERE` clause
 - `AND`, `OR`, ...,
 - Arithmetic expressions
 - String expressions using `LIKE` or `NOT LIKE`:
 - `%` implies any string, `_` implies any character



Null Values

- Tuples can have “NULL” values for some attributes
 - Either means missing (exists but don’t know what it is)
 - or inapplicable (value doesn’t apply: occupation for a child)
- Need to be careful in how to deal with NULLs.
- NULLs do not satisfy conditions:
 - $\text{length} > 100$ evaluates to FALSE for a NULL value of length
 - $\text{length} \leq 100$ evaluates to FALSE for a NULL value of length
- Thus, if we want all tuples, we need to explicitly test for NULLs
 - $\text{length} > 100 \text{ OR } \text{length} \leq 100 \text{ OR } \text{length IS NULL}$
- If you want to learn more about NULLs, look up “three-valued logic”



Multi-relation Queries

- List them in the FROM clause
- Use “As” to rename them if needed
- Like in relational algebra, refer to attributes as <relation>.<attribute> if needed
- Payment (payment_id, customer_id, staff_id, rental_id, amount, payment_date)
- Rental (rental_id, rental_date, inventory_id, customer_id, return_date, staff_id, last_update)
- SELECT * FROM Payment; SELECT * FROM Rental;
- Find customers who have paid for their rentals
 - SELECT Rental.customer_id from Rental,Payment WHERE Payment.customer_id = Rental.customer_id AND Payment.rental_id=Rental.rental_id;
 - Can find distinct customers using DISTINCT



Another Example

- Film_actor (actor_id, film_id, last_update), Actor (actor_id, first_name, last_name, last_update), Film (film_id, ...)
- Find actors in the movie “Caddyshack Jedi”
 - SELECT Actor.first_name, Actor.last_name
 - FROM Film, Actor, Film_actor
 - WHERE Film.film_id = Film_actor.film_id
 - AND Actor.actor_id = Film_actor.actor_id
 - AND Film.title = 'Caddyshack Jedi'



How to Read a SQL Query

- Start with the FROM clause: lists the tables
 - Imagine that we are considering every combination of tuples in the tables via a cartesian product
 - For each combination do the following steps:
- Apply the selection in the WHERE clause
- Apply the projection in the SELECT clause
- Convention: keywords caps, attribs small, rel first name caps



Another Example: a “Self-Join”

- Find actors who share the same first name but different last names
 - `SELECT a1.first_name, a1.last_name, a2.last_name`
 - `FROM Actor AS a1, Actor AS a2`
 - `WHERE a1.first_name = a2.first_name`
 - `AND a1.last_name != a2.last_name;`



Translation to Relational Algebra

- `SELECT a1, a2, ..., FROM R1, R2, ..., WHERE <cond>`



Translation to Relational Algebra

- SELECT a_1, a_2, \dots , FROM R_1, R_2, \dots , WHERE $\langle \text{cond} \rangle$

$$\Pi_{a_1, a_2, \dots}(\sigma_{\langle \text{cond} \rangle}(R_1 \times R_2 \times \dots))$$

- Select-From-Where = Join (or Cross Product)-Select-Project
- Equivalent semantics:
 - consider every combination of tuples from $R_1 \times R_2 \times \dots$,
 - apply the condition in WHERE clause,
 - add to the output as described in the SELECT clause.



Recap

- Data-savviness is the future!
- Notion of a DBMS
- The relational data model and algebra: bags and sets
- Next: SQL
 - SFW and its semantics
 - LIKE, AS, *, %, ...
 - Null values
 - Single and multiple relations



Queries

- Any other queries you'd like to try?

