# COMP 330: SQL 1

Chris Jermaine and Kia Teymourian

Rice University

# SQL

De-facto standard DB programming language

- First proposed by IBM researchers in 1970's
- Oracle first to offer commercial version in 1979
- IBM soon after

SQL is a H U G E language!!

- Current standard runs to 100s of pages
- Consists of a declarative DML
- And an imperative DML
- And a DDL

We begin with the heart and soul of SQL: the declarative DML

# Our First Query

LIKES (DRINKER, BEER)

FREQUENTS (DRINKER, BAR)

SERVES (BAR, BEER)

Who goes to a bar serving Sam Smith Taddy Porter? ('SSTP')

# Our First Query

LIKES (DRINKER, BEER)

FREQUENTS (DRINKER, BAR)

SERVES (BAR, BEER)

Who goes to a bar serving Sam Smith Taddy Porter? ('SSTP')

```
SELECT DISTINCT f.DRINKER
FROM FREQUENTS f, SERVES s
WHERE f.BAR = s.BAR AND s.BEER = 'SSTP'
```

What happens without `DISTINCT`?

# Our First Query

LIKES (DRINKER, BEER)

FREQUENTS (DRINKER, BAR)

SERVES (BAR, BEER)

Who goes to a bar serving Sam Smith Taddy Porter? ('SSTP')

```
SELECT DISTINCT f.DRINKER
FROM FREQUENTS f, SERVES s
WHERE f.BAR = s.BAR AND s.BEER = 'SSTP'
```

What happens without `DISTINCT`?

Closely related to RC! Same as:

- $\{f.\text{DRINKER} | \text{FREQUENTS}(f) \wedge \text{SERVES}(s) \wedge f.\text{BAR} = s.\text{BAR} \wedge s.\text{BAR} = \text{'SSTP'}\}$

# Subqueries

Can have a subquery in the `WHERE` clause

Linked with keywords

- `EXISTS`
- `IN`
- `ALL`
- `SOME`

# Subquery Example

LIKES (DRINKER, BEER)

FREQUENTS (DRINKER, BAR)

SERVES (BAR, BEER)

Who likes all of the beers that Chris likes?

# Subquery Example

LIKES (DRINKER, BEER)

FREQUENTS (DRINKER, BAR)

SERVES (BAR, BEER)

Who likes all of the beers that Chris likes?

```
SELECT l.DRINKER
FROM LIKES l
WHERE NOT EXISTS  (a beer Chris likes that is not
    also liked by l.DRINKER)
```

# Subquery Example

LIKES (DRINKER, BEER)

FREQUENTS (DRINKER, BAR)

SERVES (BAR, BEER)

Who likes all of the beers that Chris likes?

```
SELECT l.DRINKER
FROM LIKES l
WHERE NOT EXISTS (
  SELECT l2.BEER
  FROM LIKES l2
  WHERE l2.DRINKER = 'Chris' AND l2.BEER NOT IN (
    the set of beers liked by l.DRINKER))
```

# Subquery Example

LIKES (DRINKER, BEER)

FREQUENTS (DRINKER, BAR)

SERVES (BAR, BEER)

Who likes all of the beers that Chris likes?

```
SELECT l.DRINKER
FROM LIKES l
WHERE NOT EXISTS (
  SELECT l2.BEER
  FROM LIKES l2
  WHERE l2.DRINKER = 'Chris' AND l2.BEER NOT IN (
    SELECT l3.beer
    FROM LIKES l3
    WHERE l3.DRINKER = l.DRINKER))
```

# Subquery Example

```
SELECT l.DRINKER
FROM LIKES l
WHERE NOT EXISTS (
   SELECT l2.BEER
   FROM LIKES l2
   WHERE l2.DRINKER = 'Chris' AND l2.BEER NOT IN (
      SELECT l3.beer
      FROM LIKES l3
      WHERE l3.DRINKER = l.DRINKER))
```

Same as:

- $\{l.\text{DRINKER}|\text{LIKES}(l) \wedge \text{not}\exists(l_2)(\text{LIKES}(l_2) \wedge l_2.\text{DRINKER} = \text{'Chris'} \wedge \text{not}\exists(l_3)(\text{LIKES}(l_3) \wedge l_3.\text{DRINKER} = l.\text{DRINKER} \wedge l_3.\text{BAR} = l_2.\text{BAR}))\}$

# SOME predicate

RATES (DRINKER, BEER, SCORE)

SOME is used like "`expression boolOp SOME (subquery)`"

returns true if some item in the subquery can make the boolOp eval to true

Ex: List the beers that are not Chris' favorite.

# SOME predicate

RATES (DRINKER, BEER, SCORE)

SOME is used like "`expression boolOp SOME (subquery)`"

returns true if some item in the subquery can make the boolOp eval to true

Ex: List the beers that are not Chris' favorite.

```
SELECT r.BEER
FROM RATES r
WHERE r.DRINKER = 'Chris' AND r.SCORE <= SOME (
   SELECT r2.SCORE
   FROM RATES r2
   WHERE r.DRINKER = 'Chris')
```

- ALL is similar!
- but boolOp must eval to true for everything in subquery

# Some Closing Notes

## Style

- Declarative SQL codes tend to be very short
- Good because effort, bugs $\propto$ code length
- Bad because sometimes difficult to understand!

# Some Closing Notes

## Style

- Declarative SQL codes tend to be very short
- Good because effort, bugs $\propto$ code length
- Bad because sometimes difficult to understand!

## Hence, style is important. Some suggestions

- Always alias tuple variables
- Always indent carefully
- Only one major keyword per line (`SELECT`, `FROM`, etc.)
- Pick a capitalization schema and religiously stick to it
- Make frequent use of views...

# Views

"Common" (non-materialized) views are just macros

Ex: List the beers that are not Chris' favorite.

```
CREATE VIEW CHRIS_BEERS AS
SELECT *
FROM RATES r
WHERE r.DRINKER = 'Chris'

SELECT r.BEER
FROM CHRIS_BEERS
WHERE r.SCORE <= SOME (
  SELECT r2.SCORE
  FROM CHRIS_BEERS)
```

# Questions?