COMP 330: SQL 1

Chris Jermaine and Kia Teymourian
Rice University
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.DRINKER | FREQUENTS(f) \land SERVES(s) \land f.BAR = s.BAR \land s.BAR = 'SSTP' \}
Subqueries

Can have a subquery in the \texttt{WHERE} clause

Linked with keywords

\begin{itemize}
  \item \texttt{EXISTS}
  \item \texttt{IN}
  \item \texttt{ALL}
  \item \texttt{SOME}
\end{itemize}
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 1.DRINKER
FROM LIKES 1
WHERE NOT EXISTS (a beer Chris likes that is not also liked by 1.DRINKER)
Subquery Example

LIKES (DRINKER, BEER)
FREQUENTS (DRINKER, BAR)
SERVES (BAR, BEER)

Who likes all of the beers that Chris likes?

```sql
SELECT 1.DRINKER
FROM LIKES 1
WHERE NOT EXISTS (  
    SELECT 12.BEER
    FROM LIKES 12
    WHERE 12.DRINKER = 'Chris' AND 12.BEER NOT IN (  
        the set of beers liked by 1.DRINKER  
    )
)
```
Subquery Example

LIKES (DRINKER, BEER)
FREQUENTS (DRINKER, BAR)
SERVES (BAR, BEER)

Who likes all of the beers that Chris likes?

SELECT 1.DRINKER
FROM LIKES 1
WHERE NOT EXISTS (SELECT 12.BEER
FROM LIKES 12
WHERE 12.DRINKER = 'Chris' AND 12.BEER NOT IN (SELECT 13.beer
FROM LIKES 13
WHERE 13.DRINKER = 1.DRINKER))
Subquery Example

```sql
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.DRINKER|LIKES(l) ∧ not∃(l2)(LIKES(l2) ∧ l2.DRINKER = 'Chris' ∧ not∃(l3)(LIKES(l3) ∧ l3.DRINKER = l.DRINKER ∧ l3.BAR = l2.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.

```sql
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...
“Common” (non-materialized) views are just macros

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

```sql
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?