COMP 330/543: SQL 1

Luis Guzman

Sinan Kockara

Chris Jermaine

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
- Donald D. Chamberlin Video

SQL is a HUGE language!!

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

Relational Calculus/Algebra vs. SQL

Duplicates are not automatically eliminated

Not all SQL implementations are compatible

- Support different set of operators
- Date and time syntax
- Comparison case sensitivity

SQL extends RC/RA

- Aggregate functions
- Schema modifications

RA vs SQL Operators

RA Name	RA symbol	SQL equivalent
Projection	π	SELECT [L: attribute list]
Join	$\times \quad \bowtie *$	FROM [R: Relation list]
Selection	σ	WHERE [C: Condition list]
Union	$\cup$	UNION or UNION ALL
Intersection	$\cap$	JOIN or EXISTS or IN
Difference	$-$	EXCEPT
Rename	ρ	AS
Assignment	$\leftarrow$	INTO

Query Structure

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

```
SELECT <attibute list>
FROM <tables>
WHERE <conditions> (Optional)
```

```
SELECT DRINKER, BEER FROM LIKES
```

DRINKER	BEER
Chris	Double Trouble
Chris	Tout Suite
Luis	Blue Moon
Luis	Modelo

Query Structure

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

```
SELECT <attibute list>
FROM <tables>
WHERE <conditions>
```

```
SELECT DRINKER, BEER FROM LIKES WHERE DRINKER = "Luis"
```

DRINKER	BEER
Luis	Blue Moon
Luis	Modelo

Our First “Real” Query

LIKES (DRINKER, BEER)

FREQUENTS (DRINKER, BAR)

SERVES (BAR, BEER)

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

```
SELECT

FROM

WHERE
```

Our First “Real” Query

LIKES (DRINKER, BEER)

FREQUENTS (DRINKER, BAR)

SERVES (BAR, BEER)

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

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

Are we missing anything?

Our First “Real” Query

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

FREQUENTS

```
(’Luis’, ’Bar1’)
(’Luis’, ’Bar2’)
```

SERVES

```
(’Bar1’, ’SSTP’)
(’Bar1’, ’Modelo’)
(’Bar2’, ’Blue␣Moon’ )
(’Bar2’, ’SSTP’ )
```

OUTPUT

```
(’Luis’)
(’Luis’)
```

Our First “Real” 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 AS f, SERVES AS s
WHERE f.BAR = s.BAR AND s.BEER = "SSTP"
```

Are we missing anything?

- The DISTINCT keyword

Our First “Real” 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 AS f, SERVES AS s
WHERE f.BAR = s.BAR AND s.BEER = "SSTP"
```

Closely related to RC! Same as:

- $\{f.\text{DRINKER}|\text{FREQUENTS}(f) \wedge \exists(s)(\text{SERVES}(s) \wedge f.\text{BAR} = s.\text{BAR} \wedge s.\text{BEER} = \text{‘SSTP’})\}$

AS Keyword

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

What does AS do?

- ▷ Rename (ρ) from Relational Algebra!
- ▷ Works on tables as well as attributes
- ▷ Actual keyword is optional
- ▷ Why bother? To create a more meaningful name

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

Select Clause

Attribute	Example
Attibute list	***d***.atr1, ***d***.atr2
All attributes	*
<table name>.*	FREQUENTS.*
<alias name>.*	*f*.*
<math equation>	$1 + 3$
<constant>	'CPA', 3
<function>	`NOW`, `CONCAT`, `COALESCE`
Eliminate duplicates	`DISTINCT`

Where Clause

- <attribute> = <value>
- <attribute> BETWEEN [value1] AND [value2]
- <attribute> IN ([value1], [value2], . . .)
- <attribute> LIKE 'SST%'
- <attribute> LIKE 'SST_'
- <attribute> IS NULL and [attribute] IS NOT NULL
- Logical combinations with AND and OR
- Mathematical functions <>, !=, >, <, . . .
- Subqueries . . .

Subqueries

Can have a subquery in the `WHERE` clause

Linked with keywords

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

Subqueries

Can have a subquery in the `WHERE` clause

Linked with keywords

- `EXISTS`
  - ▷ `EXISTS` <subquery>
  - ▷ If the subquery returns at least one tuple, the clause evaluates to TRUE
  - ▷ `NOT EXISTS`?
- `IN`
- `ALL`
- `SOME`

Subqueries

Can have a subquery in the `WHERE` clause

Linked with keywords

- `EXISTS`
- `IN`
  - <expression> `IN` <subquery>/ <expression> `NOT IN` <subquery>
  - How does `IN` work?
- `ALL`
- `SOME`

What is an expression in this context?

Subqueries

Can have a subquery in the `WHERE` clause

Linked with keywords

- `EXISTS`
- `IN`
- `ALL`
  - ▷ <expression> <boolOP> `ALL` <subquery>
  - ▷ TRUE if every item in the subquery makes the boolOp evaluate to TRUE
- `SOME`
  - ▷ <expression> <boolOP> `SOME/ANY` <subquery>
  - ▷ TRUE if some item in the subquery can make the boolOp evaluate to TRUE

Subqueries

How do subqueries work?

- As we iterate over the tuples of the outer query, the inner query is evaluated for each tuple.
- Some can be evaluated just once
    - ▷ E.g., a subquery that returns the number of BARS that are frequented
- Some require the subquery to be evaluated for every value assignment in the outer query
    - ▷ E.g., a subquery that returns the number of BARS that each DRINKER goes to
    - ▷ Correlated subqueries

Subquery Example #1: IN

LIKES(DRINKER, BEER)

Q: Who likes ‘PBR’ and ‘Corona’?

1. Figure out who likes ‘PBR’
2. Use the subquery to make sure they also like ‘Corona’

Subquery Example #1: IN

LIKES(DRINKER, BEER)

Q: Who likes ‘PBR’ and ‘Corona’?

1. Figure out who likes ‘PBR’
2. Use the subquery to make sure they also like ‘Corona’

```
SELECT DISTINCT l.DRINKER
FROM LIKES l
WHERE l.BEER = 'PBR'
```

Subquery Example #1: IN

LIKES(DRINKER, BEER)

Q: Who likes ‘PBR’ and ‘Corona’?

```
SELECT DISTINCT l.DRINKER
FROM LIKES l
WHERE l.BEER = 'PBR'
        AND l.DRINKER IN  (people who like Corona)
```

Subquery Example #1: IN

LIKES(DRINKER, BEER)

Q: Who likes ‘PBR’ and ‘Corona’?

```
SELECT DISTINCT l.DRINKER
FROM LIKES l
WHERE l.BEER = 'PBR'
        AND l.DRINKER IN (
                SELECT l2.DRINKER
                FROM LIKES l2
                WHERE l2.BEER = 'Corona')
```

What is the subquery returning?

Subquery Example #1: IN

Q: Who likes ‘PBR’ and ‘Corona’?

Many subqueries can be written as JOINS

- ▷ People find it easier to reason about it one way or the other

```
SELECT DISTINCT l.DRINKER
FROM LIKES l
WHERE l.BEER = ’PBR’
        AND l.DRINKER IN (
                SELECT l2.DRINKER
                FROM LIKES l2
                WHERE l2.BEER = ’Corona’)
```

```
SELECT DISTINCT l1.DRINKER
FROM LIKES l1, LIKES l2
WHERE l1.DRINKER = l2.DRINKER
        AND l1.BEER = ’PBR’
        AND l2.BEER = ’Corona’
```

Subquery Example #2: SOME

RATES (DRINKER, BEER, SCORE)

Q: List the beers that are not Luis' favorite.

What does it mean, in terms of RATES, when we say favorite?

Subquery Example #2: SOME

RATES (DRINKER, BEER, SCORE)

Q: List the beers that are not Luis’ favorite.

1. Find the beers that Luis likes

2. Use the subquery to select every non-favorite beer

```
SELECT r.BEER
FROM RATES r
WHERE r.DRINKER = ’Luis’
```

Subquery Example #2: SOME

RATES (DRINKER, BEER, SCORE)

Q: List the beers that are not Luis' favorite.

What does it mean, in terms of RATES, when we say favorite?

```
SELECT r.BEER
FROM RATES r
WHERE r.DRINKER = 'Luis' AND  (this beer score must not be the highest)
```

Subquery Example #2: SOME

RATES (DRINKER, BEER, SCORE)

Q: List the beers that are not Luis' favorite.

What does it mean, in terms of RATES, when we say favorite?

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

What is the subquery returning?

Views

Q: List the beers that are not Luis’ favorite.

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

- Unexecuted query
- Can be used in place of a table
- Convenient way to simplify a query
- Query is executed when view is used by another query
- Its results are not stored

```
CREATE VIEW LUIS_BEERS AS
SELECT *
FROM RATES r
WHERE r.DRINKER = ’Luis’
```

Views

Q: List the beers that are not Luis' favorite.

```
CREATE VIEW LUIS_BEERS AS
SELECT *
FROM RATES r
WHERE r.DRINKER = 'Luis'

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

Subquery Example #3: NOT EXISTS

LIKES (DRINKER, BEER)

FREQUENTS (DRINKER, BAR)

SERVES (BAR, BEER)

Q: Who likes all of the beers that Luis likes?

Subquery Example #3: NOT EXISTS

LIKES (DRINKER, BEER)

FREQUENTS (DRINKER, BAR)

SERVES (BAR, BEER)

Q: Who likes all of the beers that Luis likes?

- There does not exist a beer that Luis likes that is not also liked by these drinkers
- Every beer Luis likes is liked by these drinkers BUUUUT they might like other beers as well

Subquery Example #3: NOT EXISTS

LIKES (DRINKER, BEER)

FREQUENTS (DRINKER, BAR)

SERVES (BAR, BEER)

Q: Who likes all of the beers that Luis likes?

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

Subquery Example #3: NOT EXISTS

LIKES (DRINKER, BEER)

FREQUENTS (DRINKER, BAR)

SERVES (BAR, BEER)

Q: Who likes all of the beers that Luis likes?

1. Beer that Luis likes

```
SELECT l2.BEER
FROM LIKES l2
WHERE l2.DRINKER = ‘Luis’
```

Subquery Example #3: NOT EXISTS

LIKES (DRINKER, BEER)

FREQUENTS (DRINKER, BAR)

SERVES (BAR, BEER)

Q: Who likes all of the beers that Luis likes?

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

Subquery Example #3: NOT EXISTS

LIKES (DRINKER, BEER)

FREQUENTS (DRINKER, BAR)

SERVES (BAR, BEER)

Q: Who likes all of the beers that Luis likes?

2. Beer that l.DRINKER likes

```
SELECT l3.BEER
FROM LIKES l3
WHERE l3.DRINKER = l.DRINKER
```

Subquery Example #3: NOT EXISTS

Q: Who likes all of the beers that Luis likes?

Putting it all together

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

Subquery Example #3: NOT EXISTS

Q: Who likes all of the beers that Luis likes?

Putting it all together

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

Same as:

- $\{l.\text{DRINKER}|\text{LIKES}(l) \wedge \neg\exists(l_2)(\text{LIKES}(l_2) \wedge \ l_2.\text{DRINKER} = \text{'Luis'}$ $\wedge \neg\exists(l_3)(\text{LIKES}(l_3) \wedge \ l_3.\text{DRINKER} = l.\text{DRINKER} \wedge l_3.\text{BEER} = l_2.\text{BEER}))\}$

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

Questions?