Tools & Models for Data Science

SQL 1

Chris Jermaine & Risa Myers

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

- 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

Relational Algebra vs. SQL

- Duplicates are not automatically eliminated (Multi-sets vs. sets)
- Not all SQL implementations support all RA operators
  - e.g. Difference operator
- SQL extends RA
  - with aggregate functions
  - with schema modifications

Relational Algebra Operators to SQL

RA Name	RA symbol	SQL term
Projection	π	SELECT [L: attribute list]
Join	$\times$ $\bowtie$ $*$	FROM [R: Relation list]
Selection	σ	WHERE [C: Condition list]

$$\pi_L(\sigma_C(R))$$

Relational Algebra Operators to SQL

RA Name	RA Symbol	SQL equivalent
Union	$\cup$	UNION or UNION ALL
Intersection	$\cap$	JOIN or EXISTS or IN
Difference	$-$	JOIN with NULL test or EXCEPT
Rename	ρ	AS
Assignment	$\leftarrow$	INTO

Our First Query

- Given the following relations:

LIKES (DRINKER, COFFEE)
FREQUENTS (DRINKER, CAFE)
SERVES (CAFE, COFFEE)

- Who goes to a cafe serving Cold Brew?

```
SELECT

FROM

WHERE
```

Our First Query

- Given the following relations:

  LIKES (DRINKER, COFFEE)
  FREQUENTS (DRINKER, CAFE) SERVES (CAFE, COFFEE)

- Who goes to a cafe serving Cold Brew?

```
SELECT DISTINCT f.DRINKER
FROM FREQUENTS AS f, SERVES AS s
WHERE f.CAFE = s.CAFE AND s.COFFEE = 'Cold␣Brew'
```

? What happens without `DISTINCT`?

Our First Query

LIKES (DRINKER, COFFEE)
FREQUENTS (DRINKER, CAFE)
SERVES (CAFE, COFFEE)

- Who goes to a cafe serving Cold Brew?

```
SELECT DISTINCT f.DRINKER
FROM FREQUENTS AS f, SERVES AS s
WHERE f.CAFE = s.CAFE AND s.COFFEE = 'Cold␣Brew'
```

- Closely related to RC! Same as:

$$\{f.\text{DRINKER} | \text{FREQUENTS}(f) \wedge \text{SERVES}(s) \\ \wedge f.\text{CAFE} = s.\text{CAFE} \wedge s.\text{COFFEE} = \text{'Cold Brew'}\}$$

AS

```
SELECT DISTINCT f.DRINKER
FROM FREQUENTS AS f, SERVES AS s
WHERE f.CAFE = s.CAFE AND s.COFFEE = 'Cold␣Brew'
```

- What does AS do?
  - Rename (ρ) from Relational Algebra!
  - Works on tables as well as attributes
  - Actual key word is optional
  - Why bother? To create a more meaningful name

```
SELECT DISTINCT f.DRINKER "Best␣Customers"
FROM FREQUENTS  f, SERVES  s
WHERE f.CAFE = s.CAFE AND s.COFFEE = 'Cold␣Brew'
```

JOIN

```
SELECT f.*
FROM FREQUENTS AS f, SERVES AS s
WHERE f.CAFE = s.CAFE AND s.COFFEE = 'Cold Brew'
```

- What kind of join is this?
  - A Cartesian product / Cross join
  - B Theta join
  - C Natural join

SELECT-FROM-WHERE

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

SELECT f.*
FROM FREQUENTS f
```

DRINKER	CAFE
Chris	Double Trouble
Chris	Tout Suite
Risa	Java Lava
Risa	Double Trouble

SELECT Clauses

Attribute	Example	Explanation
Attibute list	d.lastName, d.firstName	Only the specified attributes
*	*	All attributes from all relations
<table name>.*	FREQUENTS.*	All the attributes from relation <table name>
<alias name>.*	f.*	All the attributes from the relation aliased to f
<math equation>	1 + 3	Evaluates the expression
<constant>	'CPA' 3	Returns the specified constant

More SELECT Clauses

Attribute	Example	Explanation
<function>	`NOW()`	Current datetime
	`CONCAT`(<attribute and string constants)	Concatenates the values
	`COALESCE`(<attributes and constants>)	Returns the first non-NULL argument
`DISTINCT`		Eliminates duplicates

CONCAT

DRINKER(FIRSTNAME, LASTNAME, DATEOFBIRTH)

```
SELECT CONCAT(firstName, '␣', lastName) AS name
FROM DRINKER
```

Also

```
SELECT firstName ||  '␣' || lastName AS name
FROM DRINKER
```

COALESCE

- Returns the first non-NULL value in the list

```
COALESCE(FIELDA, FIELDB, 'UNKNOWN')
```

CASE

DRINKER(FIRSTNAME, LASTNAME, DATEOFBIRTH)

```
SELECT CONCAT(firstName, '␣', lastName),
        CASE WHEN dateOfBirth IS NULL THEN 'Unknown' ELSE dateOfBirth::VARCHAR
        AS checkOver21
FROM Drinker
```

COURSE(CRN, COURSENAME, ROOMID)

```
SELECT crn,
        CASE crn
                WHEN 16671 THEN 'Grad␣Databases'
                WHEN 16670 THEN 'Undergrad␣Databases'
                ELSE 'Unimportant'
        END  AS "Course␣Type"
FROM Course
```

FROM Clause

1. List Relation(s)/Table(s)/View(s)
2. Specify how they are related
3. Be explicit!
   - (otherwise you get the Cartesian Product / Cross Join)

INNER JOIN

R `INNER JOIN` S `ON` R.<att> = S.<att>
R `JOIN` S `ON` R.<att> = S.<att>
R `NATURAL JOIN` S

- Used to match up tuples from different relations
- Includes only the relations with matching attribute values

Example: Inner Join

COURSE (<u>CRN</u>, NAME)
ENROLL (<u>NETID</u>, <u>CRN</u>)
STUDENT (<u>NETID</u>)

STUDENT

NETID	NAME
rbm2	Risa
abc1	Andre
bcd2	Betty
cde4	Chris

ENROLL

NETID	CRN
abc1	123
abc1	345
cde4	123

COURSE

CRN	NAME
123	COMP 430
234	COMP 533
345	COMP 530

? Which students have enrolled in a course?

Example: Inner Join

COURSE (<u>CRN</u>, NAME)
ENROLL (<u>NETID</u>, <u>CRN</u>)
STUDENT (<u>NETID</u>, <u>NAME</u>)

- Which students have enrolled in a course?
- STUDENT $\bowtie_{NETID=NETID}$ ENROLL

```
SELECT *
FROM STUDENT s INNER JOIN ENROLL e ON s.NETID = e.NETID
```

RESULTS

NETID	NAME	NETID	CRN
abc1	Andre	abc1	123
abc1	Andre	abc1	345
cde4	Chris	cde4	123

? How is a natural join different?

Example: Natural Join

COURSE (<u>CRN</u>, NAME)
ENROLL (<u>NETID</u>, <u>CRN</u>)
STUDENT (<u>NETID</u>, <u>NAME</u>)

- Which students have enrolled in a course?
- STUDENT * ENROLL

```
SELECT *
FROM STUDENT s NATURAL JOIN ENROLL e
```

RESULTS

NETID	NAME	CRN
abc1	Andre	123
abc1	Andre	345
cde4	Chris	123

Left / Right Outer Join

R `LEFT OUTER JOIN` S `ON` R.<att> = S.<att>
R `RIGHT OUTER JOIN` S `ON` R.<att> = S.<att>

- Used to match up tuples from different relations
- Includes all the relations from the "outer" side
- If there is no matching tuple, assigns NULLs
- Returns a relation with all the attributes of R • all the attributes of S
- Tip: Pick one direction and use it consistently

Example: Left Outer Join

COURSE (<u>CRN</u>, NAME)
ENROLL (<u>NETID</u>, <u>CRN</u>)
STUDENT (<u>NETID</u>, <u>NAME</u>)

STUDENT

NETID	NAME
rbm2	Risa
abc1	Andre
bcd2	Betty
cde4	Chris

ENROLL

NETID	CRN
abc1	123
abc1	345
cde4	123

COURSE

CRN	NAME
123	COMP 430
234	COMP 533
345	COMP 530

? Which students haven't enrolled in any courses?

Example: Left Outer Join

COURSE (<u>CRN</u>, NAME)
ENROLL (<u>CRN</u>, <u>NETID</u>)
STUDENT (<u>NETID</u>, <u>NAME</u>)

- Which students haven't enrolled in any courses?
- STUDENT $⟕_{NETID=NETID}$ ENROLL

```
SELECT *
FROM STUDENT s LEFT OUTER JOIN ENROLL e ON s.NETID = e.NETID
WHERE e.CRN IS NULL
```

RESULTS

NETID	NAME	NETID	CRN
rbm2	Risa	NULL	NULL
bcd2	Betty	NULL	NULL

Example: Right Outer Join

COURSE (<u>CRN</u>, NAME)
ENROLL (<u>NETID</u>, <u>CRN</u>)
STUDENT (<u>NETID</u>, <u>NAME</u>)

- ? What question does this query answer?
- ENROLL $\bowtie_{NETID=NETID}$ COURSE

```
SELECT *
FROM ENROLL e RIGHT OUTER JOIN COURSE c ON e.CRN = c.CRN
WHERE e.CRN IS NULL
```

Example: Right Outer Join

COURSE (CRN, NAME)
ENROLL (NETID, CRN)
STUDENT (NETID, NAME)

- What question does this query answer?
- ENROLL $\rtimes_{NETID=NETID}$ COURSE

```
SELECT *
FROM ENROLL e RIGHT OUTER JOIN COURSE c ON e.CRN = c.CRN
WHERE e.CRN IS NULL
```

RESULTS

NETID	CRN	CRN	NAME
NULL	NULL	234	COMP 533

Full Outer Join

R `FULL OUTER JOIN` S `ON` R.<att> = S.<att>

$R ⟗_{R.<att>=S.<att>} S$

- Used to match up tuples from different relations
- Includes all the relations from both sides
- If there is no matching tuple, assigns NULLs
- Returns a relation with all the attributes of R • all the attributes of S

Example: Full Outer Join

STUDENT (<u>NETID</u>, NAME)
TEAM (<u>TEAMNAME</u>, CAPTAINNETID)

STUDENT

NETID	NAME
ghi8	Gary
hij2	Holly
ijk12	Isabel

TEAM

TEAMNAME	CAPTAINNETID
Peanut butter	ghi8
Jelly	NULL

```
SELECT s.NAME, t.TEAMNAME
FROM STUDENT s FULL OUTER JOIN TEAM t ON s.NETID = t.CAPTAINNETID
```

? What does this expression represent?

? How might it be useful?

Example: Full Outer Join

STUDENT (<u>NETID</u>, NAME)
TEAM (<u>TEAMNAME</u>, CAPTAINNETID)

```
SELECT s.NAME, t.TEAMNAME
FROM STUDENT s FULL OUTER JOIN TEAM t ON s.NETID = t.NETID
```

RESULT

NAME	NAME
Gary	Peanut butter
NULL	Jelly
Holly	NULL
Isabel	NULL

Self Join

```
R AS R1 JOIN R AS R2 ON R1.<att> = R2.<att>
```

- Used to match up tuples from relation R back to itself
- Any type of JOIN may be used
- Returns a relation with all the attributes of R • all the attributes of R

Example: Self Join

FACULTY (NETID, NAME, MGRNETID)

FACULTY

NETID	NAME	MGRNETID
rbm2	Risa	abc1
abc1	Andre	bcd2
bcd2	Betty	abc1
cde4	Chris	NULL

```
SELECT f.NAME, Mgr.Name
FROM FACULTY f JOIN FACULTY Mgr ON f.MGRNETID = Mgr.NETID
```

? What does this expression represent?

A Every faculty member paired with every manager

B Every faculty member who has. a manager, paired with that manager

Example: Self Join

FACULTY (<u>NETID</u>, NAME, MGRNETID)

FACULTY

NETID	NAME	MGRNETID
rbm2	Risa	bcd2
abc1	Andre	bcd2
bcd2	Betty	cde4
cde4	Chris	NULL

```
SELECT f.NAME, Mgr.Name
FROM FACULTY f JOIN FACULTY Mgr ON f.MGRNETID = Mgr.NETID
```

RESULT

NAME	NAME
Risa	Betty
Andre	Betty
Betty	Chris

True / False Questions

1. SQL queries typically start with a SELECT clause
2. SELECT clauses must contain at least one attribute
3. Every LEFT JOIN expression can be written as a RIGHT JOIN expression
4. A Cartesian Product is also known as a CROSS JOIN or CROSS PRODUCT
5. SELF JOINs run the query twice
6. SELF JOINs use the same table more than once
7. SQL statements ignore whitespace
8. Attributes can be renamed to "user friendly" labels

Wrap up

- How can we use what we learned today?
- What do we know now that we didn't know before?