Carnegie Mellon Univ.
Dept. of Computer Science
15-415/615 - DB Applications

C. Faloutsos – A. Pavlo

Lecture#7: Fun with SQL (Part 2)

Last Class

- Basic Queries
- Table Definition (DDL)
- NULLs
- String/Date/Time/Set/Bag Operations
- Output Redirection/Control

Today's Class: OLAP

- Views
- Joins
- Aggregations + Group By
- Nested Queries
- Window Functions
- Common Table Expressions

Example Database

STUDENT

sid	name	login	age	gpa
53666	Kayne	kayne@cs	39	4.0
53688	Bieber	jbieber@cs	22	3.9
53655	Tupac	shakur@cs	26	3.5

ENROLLED

sid	cid	grade
53666	15-415	C
53688	15-721	A
53688	15-826	B
53655	15-415	B
53666	15-721	C

COURSE

cid	name
15-415	Database Applications
15-721	Database Systems
15-826	Data Mining
15-823	Advanced Topics in Databases

Views

- Creates a "virtual" table containing the output from a **SELECT** query.
- Mechanism for hiding data from view of certain users.
- Can be used to simplify a complex query that is executed often.
  - *Won't make it faster though!*

View Example

- Create a view of the CS student records with just their id, name, and login.

```
CREATE VIEW CompSciStudentInfo AS
  SELECT sid, name, login
    FROM student
  WHERE login LIKE '%@cs';
```

Original Table

sid	name	login	age	gpa
53666	Kayne	kayne@cs	45	4.0
53688	Bieber	jbieber@cs	21	3.9

View

sid	name	login
53666	Kayne	kayne@cs
53688	Bieber	jbieber@cs

View Example

- Create a view with the average age of the students enrolled in each course.

```
CREATE VIEW CourseAge AS
  SELECT cid, AVG(age) AS avg_age
    FROM student, enrolled
   WHERE student.sid = enrolled.sid
GROUP BY enrolled.cid;
```


View Example

- Create a view with the average age of the students enrolled in each course.

```
CREATE VIEW CourseAge AS
  SELECT cid, AVG(age) AS avg_age
    FROM student, enrolled
   WHERE student.sid = enrolled.sid
GROUP BY enrolled.cid;
```

cid	avg_age
15-415	45.0
15-721	45.0
15-826	21.0

Views vs. SELECT INTO

```
CREATE VIEW AvgGPA AS
  SELECT AVG(gpa) AS avg_gpa FROM student
   WHERE login LIKE '%@cs'
```

```
SELECT AVG(gpa) AS avg_gpa INTO AvgGPA
  FROM student WHERE login LIKE '%@cs'
```


Views vs. SELECT INTO

```
CREATE VIEW AvgGPA AS
  SELECT AVG(gpa) AS avg_gpa FROM student
   WHERE login LIKE '%@cs'
```

```
SELECT AVG(gpa) AS avg_gpa INTO AvgGPA
  FROM student WHERE login LIKE '%@cs'
```

- **INTO**→Creates static table that does not get updated when student gets updated.
- **VIEW**→Dynamic results are only materialized when needed.

Materialized Views

- Creates a view containing the output from a **SELECT** query that is automatically updated when the underlying tables change.

```
CREATE MATERIALIZED VIEW AvgGPA AS
  SELECT AVG(gpa) AS avg_gpa FROM student
   WHERE login LIKE '%@cs'
```


Today's Class: OLAP

- Views
- Joins
- Aggregations + Group By
- Nested Queries
- Window Functions
- Common Table Expressions

Join Query Grammar

```
SELECT ...
  FROM table-name1 join-type table-name2
    ON qualification
[WHERE ...]
```

- **Join-Type**: The type of join to compute.
- **Qualification**: Expression that determines whether a tuple from table1 can be joined with table2. Comparison of attributes or constants using operators =, ≠, <, >, ≤, and ≥.

INNER JOIN

sid	name	login	age	gpa
53666	Kayne	kayne@cs	39	4.0
53688	Bieber	jbieber@cs	22	3.9
53655	Tupac	shakur@cs	26	3.5
53800	Drake	drake@cs	29	3.7

sid	cid	grade
53666	15-415	C
53688	15-721	A
53688	15-826	B
53655	15-415	C
53666	15-721	C

```
SELECT name, cid, grade
  FROM student, enrolled
 WHERE student.sid = enrolled.sid
```

```
SELECT name, cid, grade
  FROM student INNER JOIN enrolled
    ON student.sid = enrolled.sid
```

```
SELECT name, cid, grade
  FROM student NATURAL JOIN enrolled
```


OUTER JOIN

sid	name	login	age	gpa
53666	Kayne	kayne@cs	39	4.0
53688	Bieber	jbieber@cs	22	3.9
53655	Tupac	shakur@cs	26	3.5
53800	Drake	drake@cs	29	3.7

sid	cid	grade
53666	15-415	C
53688	15-721	A
53688	15-826	B
53655	15-415	C
53666	15-721	C

```
SELECT name, cid, grade
  FROM student LEFT OUTER JOIN enrolled
    ON student.sid = enrolled.sid
```

name	cid	grade
Kayne	15-415	C
Bieber	15-721	A
Bieber	15-826	B
Tupac	15-415	C
Kayne	15-721	C
Drake	NULL	NULL

OUTER JOIN

sid	name	login	age	gpa
53666	Kayne	kayne@cs	39	4.0
53688	Bieber	jbieber@cs	22	3.9
53655	Tupac	shakur@cs	26	3.5
53800	Drake	drake@cs	29	3.7

sid	cid	grade
53666	15-415	C
53688	15-721	A
53688	15-826	B
53655	15-415	C
53666	15-721	C

```
SELECT name, cid, grade
  FROM enrolled RIGHT OUTER JOIN student
    ON student.sid = enrolled.sid
```

name	cid	grade
Kayne	15-415	C
Bieber	15-721	A
Bieber	15-826	B
Tupac	15-415	C
Kayne	15-721	C
Drake	NULL	NULL

Join Types

```
SELECT * FROM A JOIN B ON A.id = B.id
```

Join Type	Description
INNER JOIN	Join where A and B have same value
LEFT OUTER JOIN	Join where A and B have same value AND where only A has a value
RIGHT OUTER JOIN	Join where A and B have same value AND where only B has a value
FULL OUTER JOIN	Join where A and B have same value AND where A or B have unique values
CROSS JOIN	Cartesian Product

Today's Class: OLAP

- Views
- Joins
- Aggregations + Group By
- Nested Queries
- Window Functions
- Common Table Expressions

Aggregates

- Functions that return a single value from a bag of tuples:
  - `AVG(col)` → Return the average col value.
  - `MIN(col)` → Return minimum col value.
  - `MAX(col)` → Return maximum col value.
  - `SUM(col)` → Return sum of values in col.
  - `COUNT(col)` → Return # of values for col.

Aggregates

- Functions can only be used in the **SELECT** attribute output list.
- ***Get the # of students with a "@cs" login:***

```
SELECT COUNT(login) AS cnt
  FROM student WHERE login LIKE '%@cs'
```

Aggregates

- Functions can only be used in the **SELECT** attribute output list.
- ***Get the # of students with a "@cs" login:***

```
SELECT COUNT(login) AS cnt
  FROM student WHERE login LIKE '%@cs'
```

cnt
12

Aggregates

- Functions can only be used in the **SELECT** attribute output list.
- ***Get the # of students with a "@cs" login:***

```
SELECT COUNT(login) AS cnt
  FROM student WHERE login LIKE '%@cs'
```

cnt
12

```
SELECT COUNT(*) AS cnt
  FROM student WHERE login LIKE '%@cs'
```


Aggregates

- Can use multiple functions together at the same time.
- ***Get the number of students and their average GPA that have a "@cs" login.***

```
SELECT AVG(gpa), COUNT(sid)
  FROM student WHERE login LIKE '%@cs'
```


Aggregates

- Can use multiple functions together at the same time.
- ***Get the number of students and their average GPA that have a "@cs" login.***

```
SELECT AVG(gpa), COUNT(sid)
  FROM student WHERE login LIKE '%@cs'
```

AVG(gpa)	COUNT(sid)
3.25	12

Aggregates

- **COUNT**, **SUM**, **AVG** support **DISTINCT**
- ***Get the number of unique students that have an "@cs" login.***

```
SELECT COUNT(DISTINCT login)
  FROM student WHERE login LIKE '%@cs'
```


Aggregates

- **COUNT**, **SUM**, **AVG** support **DISTINCT**
- ***Get the number of unique students that have an "@cs" login.***

```
SELECT COUNT(DISTINCT login)
  FROM student WHERE login LIKE '%@cs'
```

COUNT(DISTINCT login)
10

Aggregates

- Output of other columns outside of an aggregate is undefined:

```
SELECT AVG(s.gpa), e.cid
  FROM enrolled AS e, student AS s
 WHERE e.sid = s.sid
```


Aggregates

- Output of other columns outside of an aggregate is undefined:

```
SELECT AVG(s.gpa), e.cid
  FROM enrolled AS e, student AS s
 WHERE e.sid = s.sid
```

AVG(s.gpa)	e.cid
3.5	???

- *Unless…*

GROUP BY

- Project tuples into subsets and calc aggregates against each subset.

```
SELECT AVG(s.gpa), e.cid
  FROM enrolled AS e, student AS s
 WHERE e.sid = s.sid
 GROUP BY e.cid
```


GROUP BY

- Project tuples into subsets and calc aggregates against each subset.

```
SELECT AVG(s.gpa), e.cid
  FROM enrolled AS e, student AS s
 WHERE e.sid = s.sid
 GROUP BY e.cid
```

e.sid	s.sid	s.gpa	e.cid
53435	53435	2.25	15-721
53439	53439	2.70	15-721
56023	56023	2.75	15-826
59439	59439	3.90	15-826
53961	53961	3.50	15-826
58345	58345	1.89	15-415

AVG(s.gpa)	e.cid
2.46	15-721
3.39	15-826
1.89	15-415

GROUP BY

- Non-aggregated values in **SELECT** output clause must appear in **GROUP BY** clause.

```
SELECT AVG(s.gpa), e.cid, s.name
  FROM enrolled AS e, student AS s
 WHERE e.sid = s.sid
 GROUP BY e.cid
```


GROUP BY

- Non-aggregated values in **SELECT** output clause must appear in **GROUP BY** clause.

```
SELECT AVG(s.gpa), e.cid, s.name
  FROM enrolled AS e, student AS s
 WHERE e.sid = s.sid
 GROUP BY e.cid, s.name
```

HAVING

- Filters output results
- Like a **WHERE** clause for a **GROUP BY**

```
SELECT AVG(s.gpa) AS avg_gpa, e.cid
  FROM enrolled AS e, student AS s
 WHERE e.sid = s.sid
 GROUP BY e.cid
 HAVING avg_gpa > 2.75;
```


CMU SCS

HAVING

- Filters output results
- Like a **WHERE** clause for a **GROUP BY**

```
SELECT AVG(s.gpa) AS avg_gpa, e.cid
  FROM enrolled AS e, student AS s
 WHERE e.sid = s.sid
 GROUP BY e.cid
 HAVING avg_gpa > 2.75;
```

AVG(s.gpa)	e.cid
2.46	15-721
3.39	15-826
1.89	15-415

CMU SCS

HAVING

- Filters output results
- Like a **WHERE** clause for a **GROUP BY**

```
SELECT AVG(s.gpa) AS avg_gpa, e.cid
  FROM enrolled AS e, student AS s
 WHERE e.sid = s.sid
 GROUP BY e.cid
 HAVING avg_gpa > 2.75;
```

AVG(s.gpa)	e.cid
2.46	15-721
3.39	15-826
1.89	15-415

avg_gpa	e.cid
3.39	15-826

CMU SCS

Today's Class: OLAP

- Views
- Joins
- Aggregations + Group By
- Nested Queries
- Window Functions
- Common Table Expressions

Nested Queries

- Queries containing other queries

"outer query" *"inner query"*

- Inner query:
  - Can appear in **FROM** or **WHERE** clause

```
SELECT name FROM student WHERE
 sid IN (SELECT sid FROM enrolled)
```

Think of this as a function that returns the result of the inner query

Nested Queries

- ***Find the names of students in '15-415'***

```
SELECT name FROM student
 WHERE ...
```

"sid in the set of people that take 15-415"

Nested Queries

- ***Find the names of students in '15-415'***

```
SELECT name FROM student
 WHERE ...
   SELECT sid FROM enrolled
    WHERE cid = '15-415'
```


Nested Queries

- ***Find the names of students in '15-415'***

```
SELECT name FROM student
 WHERE sid IN (
   SELECT sid FROM enrolled
    WHERE cid = '15-415'
 )
```

Nested Queries

- **ALL** →Must satisfy expression for all rows in sub-query
- **ANY** →Must satisfy expression for at least one row in sub-query.
- **IN** → Equivalent to '**=ANY()**'.
- **EXISTS** → At least one row is returned.

- *Nested queries are difficult to optimize. Try to avoid them if possible.*

Nested Queries

- ***Find the names of students in '15-415'***

```
SELECT name FROM student
 WHERE sid = ANY(
   SELECT sid FROM enrolled
    WHERE cid = '15-415'
 )
```


Nested Queries

- ***Find student record with the highest id.***
- This won't work in **SQL-92**:

```
SELECT MAX(sid), name FROM student;
```

- Runs in **MySQL**, but you get wrong answer:

sid	name
53688	Tupac

Nested Queries

- ***Find student record with the highest id.***

```
SELECT sid, name FROM student
 WHERE ...
```

"is greater than every other sid"

Nested Queries

- ***Find student record with the highest id.***

```
SELECT sid, name FROM student
 WHERE sid  is greater than every
   SELECT sid FROM enrolled
```


Nested Queries

- ***Find student record with the highest id.***

```
SELECT sid, name FROM student
 WHERE sid => ALL(
   SELECT sid FROM enrolled
)
```

sid	name
53688	Bieber

Nested Queries

- ***Find student record with the highest id.***

```
SELECT sid, name FROM student
 WHERE sid IN (
   SELECT MAX(sid) FROM enrolled
)
```

```
SELECT sid, name FROM student
 WHERE sid IN (
   SELECT sid FROM enrolled
    ORDER BY sid DESC LIMIT 1
)
```


Nested Queries

- ***Find all courses that nobody is enrolled in.***

```
SELECT * FROM course
 WHERE ...
```

"with no tuples in the 'enrolled' table"

cid	name
15-415	Database Applications
15-721	Database Systems
15-826	Data Mining
15-823	Advanced Topics in Databases

sid	cid	grade
53666	15-415	C
53688	15-721	A
53688	15-826	B
53655	15-415	B
53666	15-721	C

Nested Queries

- ***Find all courses that nobody is enrolled in.***

```
SELECT * FROM course
 WHERE NOT EXISTS(
    tuples in the 'enrolled' table
)
```


Nested Queries

- ***Find all courses that nobody is enrolled in.***

```
SELECT * FROM course
 WHERE NOT EXISTS(
   SELECT * FROM enrolled
    WHERE course.cid = enrolled.cid
)
```


Nested Queries

- ***Find all courses that nobody is enrolled in.***

```
SELECT * FROM course
 WHERE NOT EXISTS(
   SELECT * FROM enrolled
    WHERE course.cid = enrolled.cid
)
```

cid	name
15-823	Advanced Topics in Databases

Today's Class: OLAP

- Views
- Joins
- Aggregations + Group By
- Nested Queries
- Window Functions
- Common Table Expressions

Window Functions

- Performs a calculation across a set of tuples that related to a single row.
- Like an aggregation but tuples are not grouped into a single output tuples.

```
SELECT ... FUNC-NAME(...) OVER (...)
  FROM tableName
```

Aggregation Functions
Special Functions

How to "slice" up data
Can also sort

Window Functions

- Aggregation functions:
  - Anything that we discussed earlier
- Special window functions:
  - **ROW_NUMBER()** → Number of the current row
  - **RANK()** → Order position of the current row.

```
SELECT *, ROW_NUMBER() OVER ()
  FROM enrolled
```


Window Function

- The **OVER** keyword specifies how to group together tuples when computing the window function.
- Use **PARTITION BY** to specify group.

```
SELECT *,
    ROW_NUMBER() OVER (PARTITION BY cid)
  FROM enrolled
 ORDER BY cid
```


Window Function

- You can also include an **ORDER BY** in the window grouping.

```
SELECT *,
    ROW_NUMBER() OVER (ORDER BY cid)
  FROM enrolled
 ORDER BY cid
```

Window Functions

- ***Find the student with the highest grade for each course.***

```
SELECT *, RANK() OVER (PARTITION BY cid
                       ORDER BY grade ASC)
  FROM enrolled
```


Window Functions

- ***Get the name of the student with the second highest grade for each course.***

```
SELECT * FROM (
   SELECT C.name, S.name, E.grade,
      RANK() OVER (PARTITION BY E.cid
                    ORDER BY E.grade ASC
                   ) AS grade_rank
     FROM student S, course C, enrolled E
    WHERE S.sid = E.sid
      AND C.cid = E.cid
) AS R
WHERE R.grade_rank = 2;
```


Today's Class: OLAP

- Views
- Joins
- Aggregations + Group By
- Nested Queries
- Window Functions
- Common Table Expressions

Common Table Expressions

- Provides a way to write auxiliary statements for use in a larger query.
  - *Think of it like a temp table just for one query.*
- Alternative to nested queries and views.

```
WITH cteName AS (
   SELECT 1
)
SELECT * FROM cteName
```

Common Table Expressions

- You can bind output columns to names <u>before</u> the **AS** keyword.

```
WITH cteName (col1, col2) AS (
   SELECT 1, 2
)
SELECT col1 + col2 FROM cteName
```


Common Table Expressions

- ***Find student record with the highest id that is enrolled in at least one course.***

```
WITH cteSource (maxId) AS (
   SELECT MAX(sid) FROM enrolled
)
SELECT name FROM student, cteSource
 WHERE student.sid = cteSource.maxId
```


CTEs – Recursion

- ***Print 1 to 10.***

```
WITH RECURSIVE cteSource (counter) AS (
   (SELECT 1)
   UNION ALL
   (SELECT counter + 1 FROM cteSource
     WHERE counter < 10)
)
SELECT * FROM cteSource
```

- **Postgres CTE Demo!**

Next Class

- We begin discussing storage internals.
- This material will be important for helping you pick up dates at parties.