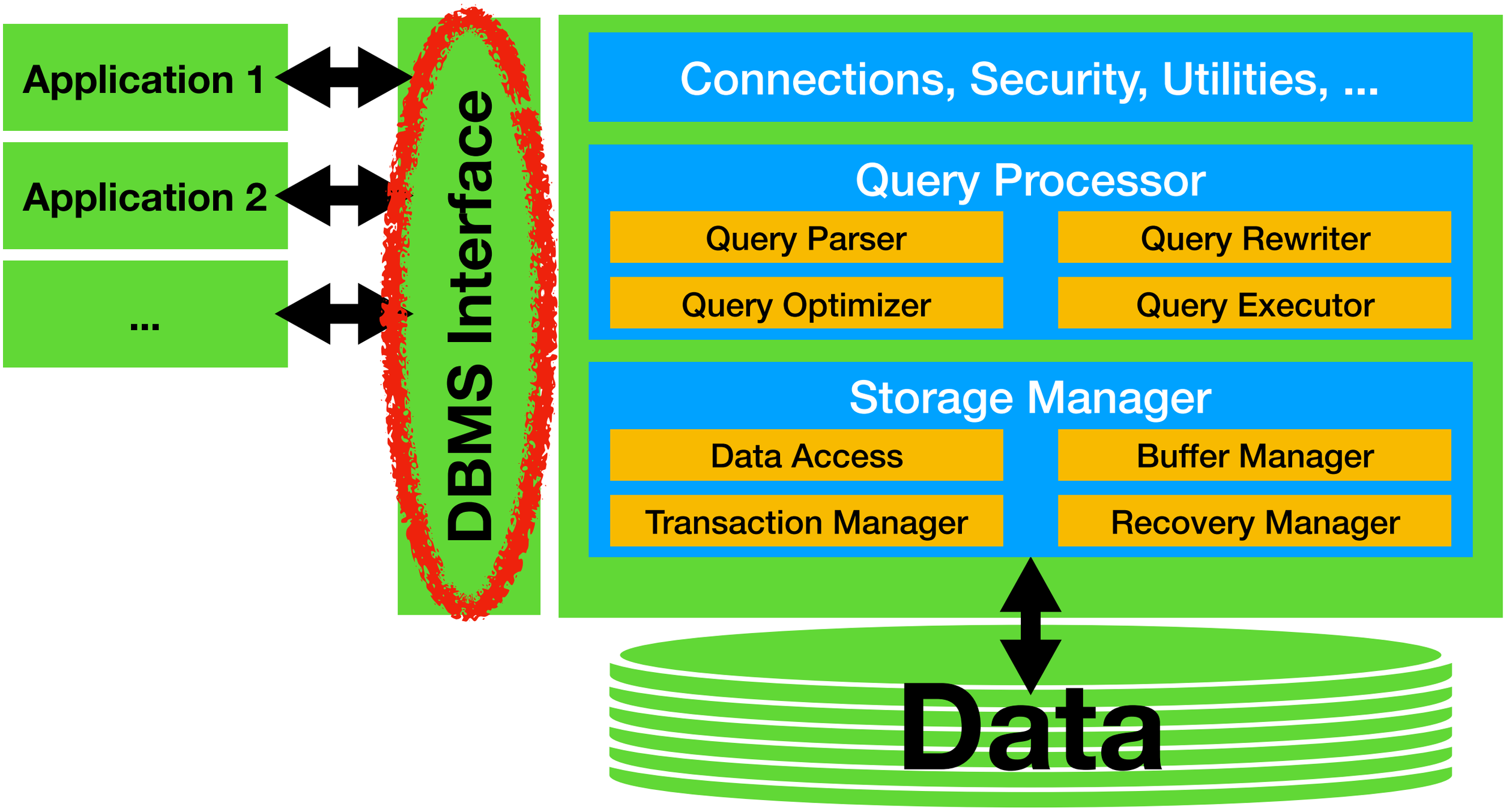
Advanced SQL Features

Immanuel Trummer
itrummer@cornell.edu
www.itrummer.org

Database Management Systems (DBMS)



Reminder: Queries So Far

SELECT …

FROM …

WHERE …

GROUP BY …

HAVING …

Two More Features

SELECT …

FROM …

WHERE …

GROUP BY …

HAVING …

ORDER BY …

LIMIT …

Syntax for Ordering

- ORDER BY **<order-item-list>**
- **<order-item>** : **<column>** **<direction>**
- **<direction>** : either **ASC** or **DESC**
- **Orders** result rows by values in order items
- Prioritize order items that appear **earlier** in list
- Applied **after** grouping (for group-by queries)
  - Items must have **unique** value per group

Limiting Output Size

- Limit **<Number>** : only shows first **<Number>** result rows

Unknown Values

- Unknown values are called **NULL** values in SQL
- SQL uses **Ternary** (i.e., Three-Valued Logic)
  - Outcome may be true, false, or unknown
- Check for corresponding outcome
  - <expression> **= TRUE**
  - <expression> **= FALSE**
  - <expression> **IS NULL** (not: "**= NULL**")
- **WHERE condition** evaluates to NULL - no result row!

Exercise (5 Minutes)

- Guess (or try in PostgreSQL) the results:
  - SELECT **3 = NULL**
  - SELECT **NULL = NULL**
  - SELECT **NULL IS NULL**
  - SELECT **NULL IS NOT NULL**
  - SELECT **TRUE OR NULL**
  - SELECT **TRUE AND NULL**

Joins with Unknowns I

- Standard join keeps only **matching** row pairs
- **Eliminates** rows without matching rows in other table
- Sometimes we want to **keep rows** regardless
- Can do that with **OUTER JOINs**
  - Fills up fields in missing row with **NULL** values

Joins with Unknowns II

- Keep each row in **left table** (plus standard join result):
  - <table-1> **LEFT OUTER JOIN** <table-2> ON ...
- Keep each row in **right table** (plus standard result):
  - <table-1> **RIGHT OUTER JOIN** <table-2> ON ...
- Keep rows in **both tables** (plus standard result):
  - <table-1> **FULL OUTER JOIN** <table-2> ON

Outer Join Example

Database Relations:

Students(Sid, Sname)
Enrollment(Sid, Cid)
Courses(Cid, Cname)

```
SELECT Sname, Count(*)
FROM Students JOIN Enrollment
    ON (Students.sid = Enrollment.sid)
GROUP BY Sname
```

Want to count number of enrollments per student

Outer Join Example

Database Relations:

Students(Sid, Sname)
Enrollment(Sid, Cid)
Courses(Cid, Cname)

Will not consider students without enrollments!

```
SELECT Sname, Count(*)
FROM Students JOIN Enrollment
    ON (Students.sid = Enrollment.sid)
GROUP BY Sname
```

Want to count number of enrollments per student

Outer Join Example

Database Relations:

Students(<u>Sid</u>, Sname)
Enrollment(<u>Sid</u>, <u>Cid</u>)
Courses(<u>Cid</u>, Cname)

Will count one row for students without enrollments!

```
SELECT Sname, Count(*)
FROM Students LEFT OUTER JOIN Enrollment
    ON (Students.sid = Enrollment.sid)
GROUP BY Sname
```

Want to count number of enrollments per student

Outer Join Example

Database Relations:

Students(Sid, Sname)
Enrollment(Sid, Cid)
Courses(Cid, Cname)

Count only students matched against courses

```
SELECT Sname, Count(cid)
FROM Students LEFT OUTER JOIN Enrollment
    ON (Students.sid = Enrollment.sid)
GROUP BY Sname
```

Want to count number of enrollments per student

Set Operations

- **Union** result tuples from two queries
  - <query-1> **UNION** <query-2> : **eliminates** duplicates
  - <query-1> **UNION ALL** <query-2> : **keep** duplicates
- **Intersect** results from two queries
  - <query-1> **INTERSECT** <query-2>
- Set **difference** between queries
  - <query-1> **EXCEPT** <query-2>
- Results from <query-1> and <query-2> must be **union-compatible**

Query Nesting

- Can use **queries as part** of another query, e.g.
  - Query instead of table in **FROM** clause,
  - Query instead of conjunct in **WHERE** clause,
  - ...
- **Query** (containing query) vs. **sub-query** (contained query)
- **Correlated** vs. **uncorrelated** sub-queries
  - Correlated sub-queries reference **containing query**

Nesting Examples

Database Relations:

Students(Sid, Sname, gpa)

(Not yet exciting)

```
SELECT SQ.Sname FROM
  (SELECT Sname FROM Students) AS SQ
```

Assign
name for sub-queries in
FROM clause

Nesting Examples

Database Relations:

Students(Sid, Sname, gpa)

```
SELECT Sname FROM Students
 WHERE gpa >= (SELECT MAX(gpa) FROM Students)
```

Sub-Queries in Conditions

- Check if sub-query result is **empty**
  - **EXISTS(<sub-query>)** : TRUE if non-empty
- Check if sub-query result **contains value**
  - **<value> IN (<sub-query>)** : TRUE if contained
- Check if condition holds **for all/some** sub-query rows
  - E.g., <value> **>= ALL(<sub-query>)** : TRUE if satisfied for all
  - E.g., <value> **>= ANY(<sub-query>)** : TRUE if satisfied for some

Nesting Examples

Database Relations:

Students(Sid, Sname, gpa)

```
SELECT Sname FROM Students
 WHERE gpa >= ALL(SELECT gpa FROM Students)
```

What does this do?

Correlated Sub-Queries

- So far: have seen **uncorrelated** sub-queries
- Uncorrelated sub-queries are a bit **"easier"**
- Correlated sub-queries: sub-query refers to **the "outside"**

Nesting Examples

Database Relations:

Students(Sid, Sname, gpa)

```
SELECT S1.Sname FROM Students S1 WHERE S1.gpa >=
  ALL(SELECT S2.gpa FROM Students S2
   WHERE S1.Sname = S2.Sname)
```

What does this do?

Evaluating Correlated Sub-Queries

- **Iterate** over rows from outer (containing) query
- Evaluate sub-query for **fixed row** in outer query
- (**Decide** whether outer row belongs into result)

Nesting Examples

Database Relations:

Students(Sid, Sname, gpa)

```
SELECT S1.Sname FROM Students S1 WHERE
EXISTS (SELECT S2.gpa FROM Students S2
WHERE S1.gpa < S2.gpa)
```

Names of all students except for students with highest gpa

Multiple Nesting Levels

Database Relations:

Students(<u>Sid</u>, Sname)
Enrollment(<u>Sid</u>, <u>Cid</u>)
Courses(<u>Cid</u>, Cname)

```
SELECT C.Cname FROM Courses C WHERE NOT EXISTS (
  SELECT * FROM Students S WHERE NOT EXISTS(
      SELECT * FROM Enrollment E
        WHERE E.cid = C.cid AND E.sid = S.sid
    )
)
```

What does this do ... ?