# CSC 365

Introduction to Database Systems

# SQL SELECT Return Values

Possible results from a SQL `SELECT`:

- **Empty set**
- **Single scalar value** - Can be included in the `SELECT` list. Also may be used for comparison in the `WHERE` clause
- **Relation / Table**
  - 1-Dimensional for `WHERE IN / ANY / ALL / EXISTS`
  - 2-Dimensional may appear in the `FROM` clause as a stand-in for a named table

# Example: Departments Established Before the Physics Dept.

```
-- find the date on which the Physics department was established
SELECT DateEstablished
FROM Department
WHERE Code = 'PHYS';


-- find departments established before the Physics department
SELECT *
FROM Department
WHERE DateEstablished <  value returned by query above;
```

# Scalar Subquery Example

```
-- Departments established before the Physics department
SELECT *
FROM Department
WHERE DateEstablished < (
  SELECT DateEstablished
  FROM Department
  WHERE Code = 'PHYS'
)
```

This subquery returns one row / one column, due to `WHERE` on primary key column.

```
-- can we do this without a subquery? (as in Lab 4)
```

# Subquery vs JOIN

Often, a query can be expressed using either a subquery or a JOIN. Another way to write the previous query:

```
SELECT D1.*
FROM Department D1
  INNER JOIN Department D2 ON D1.DateEstablished < D2.DateEstablished
WHERE D2.Code = 'PHYS'
```

- Why choose one over the other?
  - Clarity, especially in complex queries
  - Accidental row duplication or elimination with JOINs
  - Performance (based on profiling)

# WHERE: Conditions Involving Relations

Subqueries may be used with the following comparison operators in the `WHERE` clause:

- `IN`
- `ALL`
- `ANY`
- `EXISTS`

Each of these may be negated with `NOT` (`NOT IN`, `NOT EXISTS`, etc.)

# WHERE: IN / NOT IN

- `s IN R` is true if and only if *s* is equal to one of the values in *R*.
  - s may be a single value or single column name, in which case R must be a single-column relation
  - If s has more than one element, the number of elements must match the number of columns in *R*

- `s NOT IN R` is true if any only if *s* is equal to *no value* in *R*

# WHERE: IN / NOT IN Examples

```
-- Students without a minor who are in a department other than CSSE
SELECT *
FROM Student
WHERE MinorCode IS NULL
AND MajorCode NOT IN (
   SELECT Code
   FROM Discipline
   WHERE Dept = 'CSSE'
)
```

How does this query change if we use `JOIN` rather than a subquery?

# WHERE: ALL / ANY

`s <comparator> ALL R`

`s <comparator> ANY R`

(comparator may be any of the following: >, <, >=, <=, =, <>)

`s > ALL R` is true if any only if *s* is greater than every value in *unary* relation *R*. `s <> ALL R` is the same as `s NOT IN R`

`s > ANY R` is true if and only if *s* is greater than at least one value in *unary* relation *R*. `s = ANY R` is the same as `s IN R`

# WHERE: Negating ALL / ANY

`ALL` / `ANY` may be negated like other boolean expressions:

`NOT s >= ALL R` is true if and only if *s* is not the maximum value in *R*

`NOT s > ANY R` is true if any only if *s* is the minimum value in *R*

# WHERE: ALL Example

```
-- Find departments established before every department in Engineering
SELECT *
FROM Department
WHERE DateEstablished < ALL (
  SELECT DateEstablished
  FROM Department WHERE College = 'CENG'
)
```

# WHERE: ANY Example

```
-- Find the earliest-established department
SELECT *
FROM Department
WHERE NOT DateEstablished > ANY (
  SELECT DateEstablished
  FROM Department
)
```

# Nested SQL / Relational Algebra - Examples

Demo Area:

- SUPPLIERS-DEMO-2
- OUTER-JOIN-*n* exercises

# Correlated Subqueries

The subquery examples seen thus far have been uncorrelated: subqueries that can be executed once for the entire outer `SELECT`.

It is also possible to refer to values in the outer query from within a subquery, causing the subquery to be re-evaluated multiple times during execution of the outer query.

This second form of subquery is called a **correlated subquery**

# Correlated Subquery Example

```
-- List of all students, along with the number of students
-- who share their major

SELECT FirstName, LastName, MajorCode,
  (SELECT COUNT(*) FROM Student AS s2
   WHERE s2.MajorCode = s.MajorCode) as TotalInMajor
FROM Student AS s
```

Inner query references column value from outer query.

# Correlated Subquery Example 2

```
-- List sales detail, along with running total for each customer
SELECT C.Name, R.Item, R.PDate, P.Price AS ItemPrice,
  (SELECT SUM(P2.Price)
   FROM Purchases R2
   INNER JOIN Items P2 ON P2.SKU = R2.Item
   WHERE R2.Customer = R.Customer
   AND R2.PDate <= R.PDate) as CustomerRevenueToDate
FROM Purchases R
  INNER JOIN Customers C ON R.Customer = C.ID
  INNER JOIN Items P ON P.SKU = R.Item
WHERE C.Name <> 'Ben Vilasec'
ORDER BY Name, PDate
```

What does this query look in relational algebra?

# Correlated Subqueries

The previous examples demonstrate **correlated subqueries** in the `SELECT` clause

Also permitted in the `WHERE` clause:

- Direct comparisons with scalar values
- `EXISTS` operator

# WHERE: EXISTS / NOT EXISTS

- `EXISTS` *R* is true if and only if *R* is not empty
- `NOT EXISTS` *R* is true if and only if *R* is empty

# WHERE: EXISTS / NOT EXISTS Examples

```
-- Find students who are the only students in their major
SELECT *
FROM Student o
WHERE MajorCode IS NOT NULL
AND NOT EXISTS (
  SELECT StudentID
  FROM Student i
  WHERE o.MajorCode = i.MajorCode AND o.StudentID <> i.StudentID
)

-- Students who share a major with at least one other student?
```

# EXISTS / NOT EXISTS Transformation

In the WHERE clause, subqueries can be transformed into equivalent correlated subqueries using EXISTS & NOT EXISTS.

```
SELECT *
FROM Student
WHERE MajorCode IN (
  SELECT Code
  FROM Department
  WHERE College = 'OCOB'
)
```

```
SELECT *
FROM Student
WHERE EXISTS (
  SELECT Code
  FROM Department
  WHERE College = 'OCOB'
  AND Code = Student.MajorCode
)
```

# EXISTS / NOT EXISTS Transformation

List students who were enrolled on/after all departments in CENG were established.

```
SELECT *
FROM Student
WHERE DateEnrolled >= ALL (
  SELECT DateEstablished
  FROM Department
  WHERE College = 'CENG'
)
```

```
SELECT *
FROM Student
WHERE NOT EXISTS (
  SELECT DateEstablished
  FROM Department
  WHERE College = 'CENG'
  AND Student.DateEnrolled < DateEstablished
)
```

# EXISTS / NOT EXISTS Transformation

Relations R(A, B) and S(C)

```
SELECT C
FROM S
WHERE C IN (
   SELECT SUM(B)
   FROM R
   GROUP BY A
)
```

```
SELECT C
FROM S
WHERE EXISTS (
   SELECT SUM(B) FROM R
   GROUP BY A
   HAVING SUM(B) = S.C
)
```

# Using EXISTS / IN to Express Relation Difference

*[From Homework 3]*

#9: Find all bands in which 'Irmin Schmidt' DID NOT play. Output the names of the bands.

#10: Find all 'Pink Floyd' band members who did NOT participate in the recording of the album 'Meddle'. (note: a musician did not participate in a recording of an album if he did not play in the band that year).

# Conditions Involving Tuples

We've seen how `IN`, `ANY`, and `ALL` work with one-column relations. It is also possible to perform tuple-based comparisons using subqueries, as long as the *degree* matches on both sides of the comparison.

```
-- Students who share the same major & minor as another student
SELECT *
FROM Student SO
WHERE (MajorCode, MinorCode) IN (
  SELECT MajorCode, MinorCode FROM Student SI
  WHERE SI.StudentID <> SO.StudentID
)
-- Anybody missing?
```

# Nested Queries in the FROM clause

Since a `SELECT` statement returns a relational table, we can use a nested `SELECT` statement in the `FROM` clause of a query.

```
SELECT <column list>
FROM (SELECT query) [AS] <alias>
[WHERE <condition> ]
[GROUP BY <attribute list>
[HAVING <group condition>]
```

# Nested Queries in the FROM clause

Two requirements to consider when using a nested `SELECT` statement in the `FROM` clause of a query:

1. Nested `SELECT` must be enclosed in parentheses and *must have an alias*

2. All computed columns (aggregates, scalar functions, `CASE`, etc.) *must have aliases*

# Nested Queries in the FROM clause

(BAKERY dataset)

Show customer names, dates and total purchase amounts for days where the customer's total purchase amount for that day is more than two times the customer's average purchase amount on days when he/she made a purchase.

# Customer's Daily Average

Total purchases by the customer (all time) divided by the number of days on which the customer made a purchase.

```
SELECT ?
FROM BAKERY.customers C
  INNER JOIN BAKERY.receipts R ON Customer = CId
  INNER JOIN BAKERY.items ON RNumber = Receipt
  INNER JOIN BAKERY.goods ON Item = GId
GROUP BY ?
```

# Nested Queries in the FROM clause

```
SELECT C.CId, FirstName, LastName, SaleDate,
  ROUND(SUM(price), 2) AS DailyTotalPurch,
  ROUND(P.AvgDailyPurchase, 2) AS AvgDailyPurchase
FROM BAKERY.customers C
  INNER JOIN BAKERY.receipts R ON Customer = CId
  INNER JOIN BAKERY.items ON RNumber = Receipt
  INNER JOIN BAKERY.goods ON Item = GId
  INNER JOIN (
     SELECT CId, SUM(price) / COUNT(DISTINCT SaleDate) AS AvgDailyPurchase
     FROM BAKERY.customers, BAKERY.receipts, BAKERY.items, BAKERY.goods
     WHERE CId = Customer AND RNumber = Receipt AND Item = GId
     GROUP BY CId
  ) P ON  C.CId = P.CId
GROUP BY CId, FirstName, LastName, SaleDate, P.AvgDailyPurchase
HAVING SUM(price) > P.AvgDailyPurchase * 2.0;
```

# Nesting Subqueries

Subqueries can be deeply nested (up to practical limits imposed by RDBMS implementations.) Inner queries can reference outer columns, but cannot reference siblings.

```
-- Non-CENG departments with students who enrolled
-- prior to the date the department was established
SELECT *
FROM Department d1
WHERE EXISTS (
  SELECT StudentID FROM Student s
  WHERE s.MajorCode = d1.Code AND DateEnrolled < (
    SELECT DateEstablished FROM Department d2
    WHERE d2.Code = s.MajorCode
  ) AND s.MajorCode NOT IN (
    SELECT Code FROM Department d3
    WHERE d3.College = 'CENG'
  )
)
-- How would we list the corresponding students?
```

# Subquery Recap

- Considerable flexibility, deep nesting allowed

- Can appear throughout `SELECT` statement (with some exceptions)

- Subquery variations:
  - Uncorrelated
  - Derived table in `FROM` clause
  - Correlated
    - No correlation permitted for subqueries that appear `FROM` clause

- *Performance implications (profiling always recommended)*

# Control Row Count Returned by SELECT

ANSI SQL does not define a standard way to limit the number of rows returned by a `SELECT` query.

Each RDBMS has its own syntax.  In MySQL, we use `LIMIT`:

```
SELECT *
FROM Student
ORDER BY LastName
LIMIT 2
```

Sorting (`ORDER BY`) is performed **before** `LIMIT`, so we see the first two students, based on A-Z ordering of last name.

# MySQL LIMIT Variations

Single-argument `LIMIT` returns up to the specified number of rows:

```
LIMIT 5      -- top 5 rows in result set
```

Two-argument `LIMIT` returns up to the requested number of rows, after a given offset (zero indexed):

```
LIMIT 5,10   -- rows 6-15
```

# Row Limit in other RDBMSs

| **RDBMS** | **Syntax** |
|---|---|
| MySQL & PostgreSQL | `SELECT * FROM table LIMIT 10` |
| Microsoft SQL Server & Access | `SELECT TOP 10 * FROM table` |
| Oracle | `SELECT * FROM`<br>` (SELECT * FROM table)`<br>`WHERE rownum <= 10` |
| IBM DB2 | `SELECT * FROM table`<br>`FETCH FIRST 10 ROWS ONLY` |
| Informix | `SELECT FIRST 10 * FROM table` |

# Using LIMIT in Subqueries

```
SELECT Code, DeptName, DateEstablished,
  DATEDIFF((SELECT DateEstablished
            FROM Department
            ORDER BY DateEstablished DESC
            LIMIT 1),
           DateEstablished) /  365 AS YearsOlderThanNewestDept
FROM Department
```

Can we write this query without LIMIT?

# LIMIT: Use with Caution

```
-- List customer(s) with the fewest number of purchases
SELECT C.CId, C.FirstName, COUNT(*) AS PurchaseCount
FROM BAKERY.receipts R
  INNER JOIN BAKERY.customers C ON R.Customer = C.CId
GROUP BY C.Cid
ORDER BY PurchaseCount
LIMIT 1
```