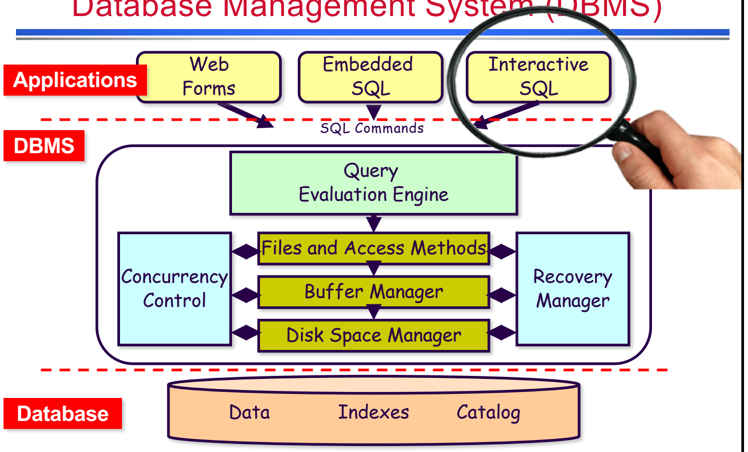
Structured Query Language
SQL - DML

- Relational Operators
- Set Relational Operators
- Retrieving with NULLs
- Nested Operations

Database Management System (DBMS)



Database Schemas for the Examples

- Student Database:

  STUDENT (SID, Name, Major, Class, QPA, Phone);
  [STUDENT (SID, FName, LName, DEPT, Class, …);]
  COURSE (CID, Name, Credits);
  ENROLLS (CID,SID,Term, Grade);
  STAFF (SSN, SID, Name, Major);

- Library Database:

  LIBRARIAN (SSN, Name, City, Gender, Salary, SNO);
  SECTION (SNO, Name, HeadSSN);
  DEPENDENT (Name, LIBSSN, DSSN, Gender, DoB);

Relational Query Languages

- **Query languages:** Allow manipulation and retrieval of data from a database
- User queries should be declarative, specifying **what** is to be retrieved; the **how** is the responsibility of the system.
  - Textual (SQL, QUEL), Graphical (QBE), Visual (QBI)
- Two mathematical Query Languages form the basis for "real" relational languages and for their implementation:
  1. **Relational Algebra:** Operational, very useful for representing execution plans.
  2. **Relational Calculus** (Tuple or Domain)**:** Lets users describe what they want, rather than how to compute it.

Declarative Query

```
SELECT   Title, TeamName
FROM     SUPERBOWL
WHERE    Rank = 1;
```

Title	TeamName
Champions	Steelers

SQL Select Statement

- Complete form:

```
SELECT [DISTINCT I ALL] attribute-list
FROM   table-list
WHERE selection-condition
GROUP BY grouping-attribute(s)
HAVING grouping-condition
ORDER BY {attribute ASC I DESC} pairs
```

Recall - Preliminaries

- A query is applied to "relation instances" (tables), and the result of a query is also a relation instance (table)
- List-oriented (positional) notation vs. Set-oriented (named-field) notation:
  - Both used in SQL-Select

SQL Insert

STUDENT(SID,Name,Major,QPA)

- **Implicit (list)**:

```
INSERT INTO STUDENT
VALUES (165, 'Susan Jones', 'CS', 0.00);
```

- **Explicit (set)**:

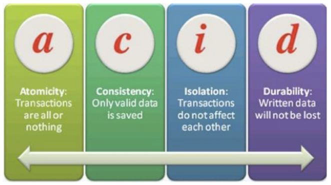
```
INSERT INTO STUDENT (SID, Name)
VALUES (165, 'Susan Jones');

INSERT INTO STUDENT (Name, SID)
VALUES ( 'Susan Jones', 165);
```

- Values-clause may be a list of tuples in some systems

Execution Abstraction

- *A **transaction** is a **logical unit of work** in DBMSs*
  - *It is the execution of a **program segment** that performs some function or task by accessing shared data (e.g., a db)*
  - logical grouping of query and update requests needed to perform a task
- Examples:
  - banking transaction
    - Deposit, withdraw, transfer $
  - airline reservation
    - reserve a seat on a flight
  - inventory transaction
    - Receive, Ship, Update


a
c
i
d
Atomicity: Transactions are all or nothing
Consistency: Only valid data is saved
Isolation: Transactions do not affect each other
Durability: Written data will not be lost

ACID Properties

- Atomicity (alias failure atomicity)

  Either all the operations associated with a transaction happen or none of them happens
- Consistency Preservation

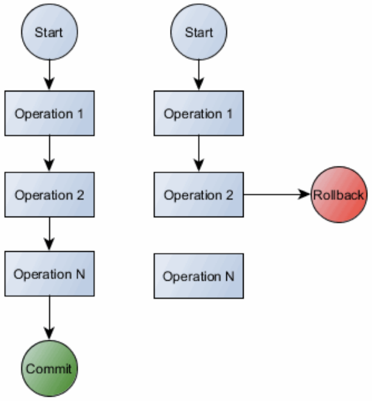
  A transaction is a correct program segment. It satisfies the integrity constraints on the database at the transaction's boundaries
- Isolation (alias concurrency atomicity / serializability)

  Transactions are independent, the result of the execution of concurrent transactions is the same as if transactions were executed serially, one after the other
- Durability (alias persistence / permanence)

  The effects of completed transactions become permanent surviving any subsequent failures

SQL TRANSACTIONS

- BEGIN
  - Each SQL statement should *implicitly* start a transaction, unless one is active.
- COMMIT [WORK]; or END [WORK];
- ROLLBACK [WORK] ;
  - ➢ ROLLBACK default action



Relational Operators in SQL

- STUDENT(<u>SID</u>, Name, Major)
- $\pi_{\text{attribute\_list}}(r)$:
  - $\pi_{\text{SID,Major}}(\text{STUDENT})$

```
SELECT  SID, Major
FROM    STUDENT;
```

- $\sigma_{\text{selection\_condition}}(r)$
  - $\sigma_{\text{Major= 'CS'}}$ (STUDENT)

```
SELECT  *
FROM    STUDENT
WHERE   Major = 'CS' ;
```

SELECT vs. WHERE

- In SQL:
  - **Selection** ***(σ)*** is expressed by the **WHERE** clause
  - **SELECT** clause actually does **Projection** ***(π)***
- It is a historical accident ☺

Basic *SQL*: Single Table Manipulation

```
SELECT    [DISTINCT|ALL] attribute-list | *
FROM      Table1
WHERE     selection-condition
```

- **DISTINCT** is an optional keyword indicating that the answer should not contain duplicates
  - Default is that duplicates are not eliminated! Why?
- *Selection-Condition:* Comparisons
  - *expression* **op** *expression*
  - **op** ∈ **{<,<=,=,>,>=,<>}**
  - combined using **AND**, **OR** and **NOT**

Aliasing in SQL: The AS Operator

- Renaming attributes in the result of a query:

  SELECT SID **AS Student_ID**
  FROM STUDENT;

- Table alias can be achieved with the AS operator in the FROM-clause: (Optional the AS)

  SELECT S.Major
  FROM STUDENT **AS S**
  WHERE S.name = 'Ruchi Agrawal';

- Renaming of attributes within a query:

  SELECT *
  FROM STUDENT **AS S(ID,FN,MJ)**
  WHERE S.FN = 'Thalia' AND S.MJ = ‘COE';

Aggregate Functions

- Tuple grouping based on the value of some attributes.

```
SELECT    List of  functions F(Attribute)
FROM      Table1
WHERE     selection-condition
```

- *F(B)* = aggregate function on attribute B
- SQL provides five aggregate functions:
  SUM, MAX, MIN, AVG, and COUNT [COUNT(□)]

Aggregate Functions… Example

- LIBRARIAN (SSN, Name, BirthDate, Gender, Salary, SNO);
- Q: Display all the statistics about librarian salaries.

```
SELECT  SUM  (Salary)  AS  TotalSalaries,
        MAX  (Salary)  AS  MaxSalary,
        MIN  (Salary)  AS  MinSalary,
        AVG  (Salary)  AS  AvgSalary,
        COUNT  (*)   AS  Cardinality,
        COUNT(DISTINCT  Salary)  AS  Salarylevels
 FROM  LIBRARIAN;
```

Note on COUNT

- COUNT (attribute-name) **does not** count NULLs
- COUNT (*) returns cardinality
- COUNT (DISTINCT attribute-name) returns the number of distinct values

Arithmetic Operator

- Arithmetic operators (+; -; *; /) may be applied on numeric values in any expression
- Q1:

```
SELECT  1.1 * SUM (Salary)
FROM    LIBRARIAN;
```

- Increment (+) and decrement (-) may be applied on data types: date, time and timestamp
- Q2:

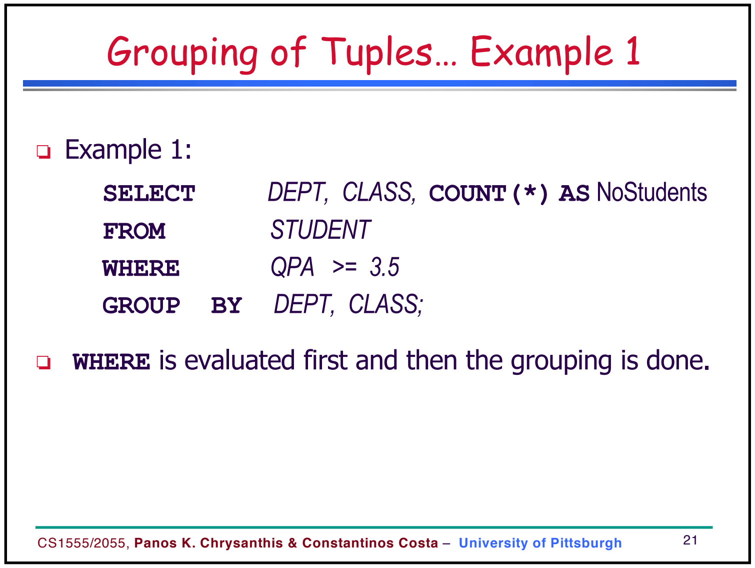
```
SELECT  Name, EXTRACT(YEAR FROM CURRENT_DATE) -
                 EXTRACT(YEAR FROM BirthDate)  AS Age
FROM    LIBRARIAN
WHERE EXTRACT(YEAR FROM CURRENT_DATE) –
          EXTRACT(YEAR FROM BirthDate) > 35;
```

Grouping of Tuples

- Tuple grouping based on the value of some attributes.

```
SELECT     A-list,  F(B)
FROM       Table1
WHERE      selection-condition
GROUP BY   A-list
HAVING     Pred
```

- ***F(B)*** = aggregate function on attribute B
- *A-list*: The grouping attributes must appear in the SELECT-clause to be meaningful
- *Pred* = a predicate on the tuples of the individual groups

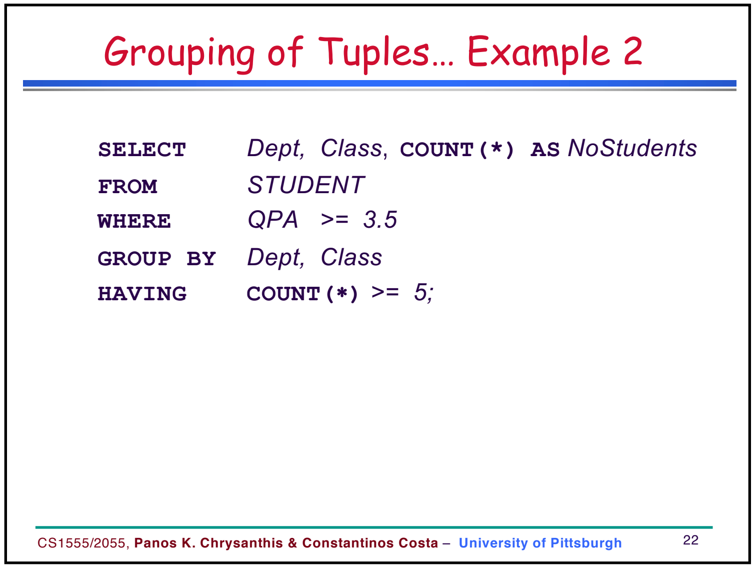


Grouping of Tuples… Example 1

- Example 1:

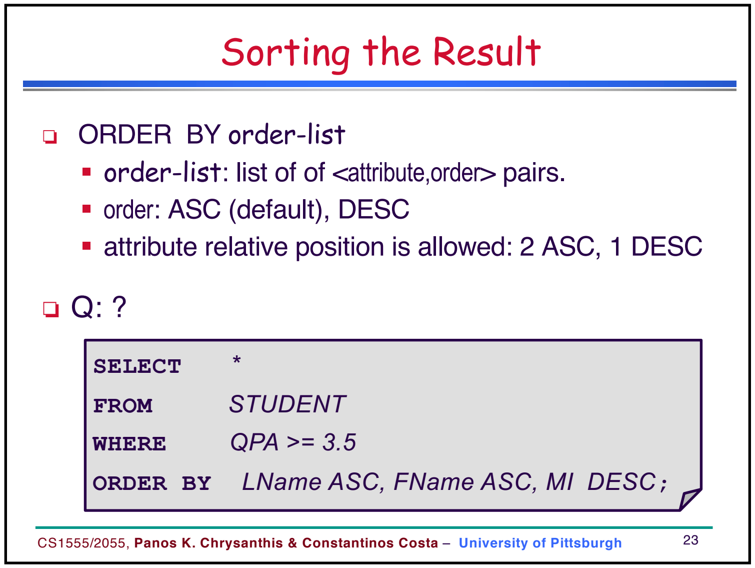
```
SELECT      DEPT, CLASS, COUNT(*) AS NoStudents
FROM        STUDENT
WHERE       QPA >= 3.5
GROUP  BY   DEPT, CLASS;
```

- **WHERE** is evaluated first and then the grouping is done.



Grouping of Tuples… Example 2

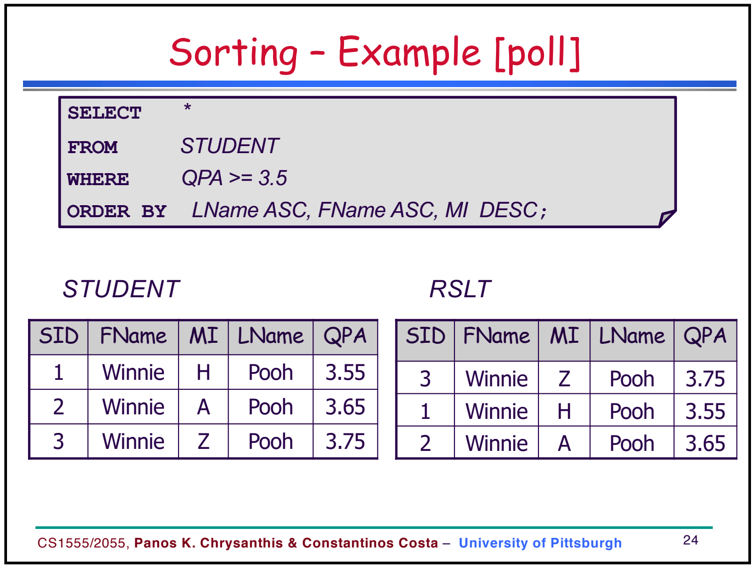
```
SELECT      Dept, Class, COUNT(*) AS NoStudents
FROM        STUDENT
WHERE       QPA  >= 3.5
GROUP BY    Dept, Class
HAVING      COUNT(*) >= 5;
```



Sorting the Result

- ORDER BY order-list
  - order-list: list of of <attribute,order> pairs.
  - order: ASC (default), DESC
  - attribute relative position is allowed: 2 ASC, 1 DESC
- Q: ?

```
SELECT     *
FROM       STUDENT
WHERE      QPA >= 3.5
ORDER BY   LName ASC, FName ASC, MI  DESC;
```



Sorting - Example [poll]

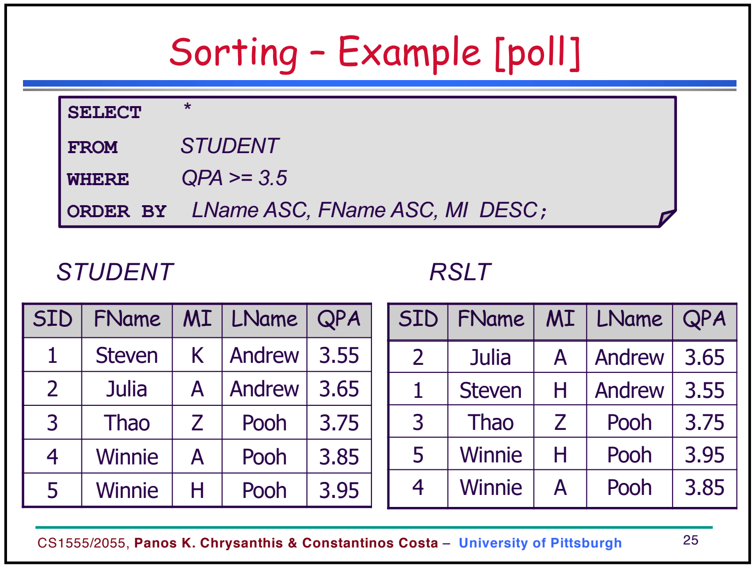
```
SELECT     *
FROM       STUDENT
WHERE      QPA >= 3.5
ORDER BY   LName ASC, FName ASC, MI  DESC;
```

STUDENT

SID	FName	MI	LName	QPA
1	Winnie	H	Pooh	3.55
2	Winnie	A	Pooh	3.65
3	Winnie	Z	Pooh	3.75

RSLT

SID	FName	MI	LName	QPA
3	Winnie	Z	Pooh	3.75
1	Winnie	H	Pooh	3.55
2	Winnie	A	Pooh	3.65



Sorting - Example [poll]

```
SELECT    *
FROM      STUDENT
WHERE     QPA >= 3.5
ORDER BY   LName ASC, FName ASC, MI  DESC;
```

STUDENT

SID	FName	MI	LName	QPA
1	Steven	K	Andrew	3.55
2	Julia	A	Andrew	3.65
3	Thao	Z	Pooh	3.75
4	Winnie	A	Pooh	3.85
5	Winnie	H	Pooh	3.95

RSLT

SID	FName	MI	LName	QPA
2	Julia	A	Andrew	3.65
1	Steven	H	Andrew	3.55
3	Thao	Z	Pooh	3.75
5	Winnie	H	Pooh	3.95
4	Winnie	A	Pooh	3.85