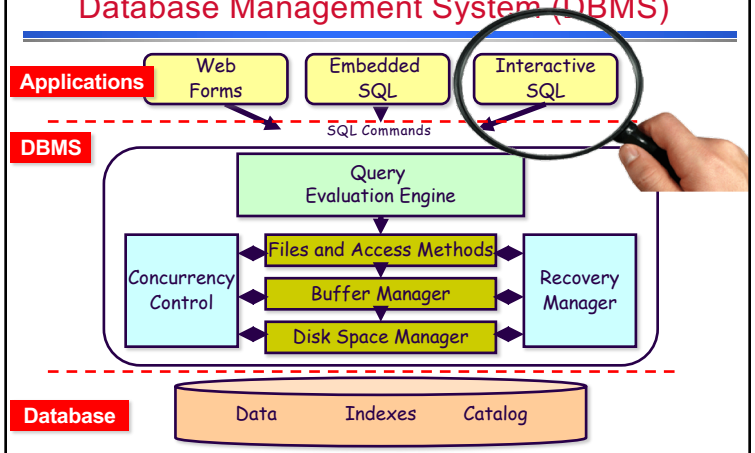
# Structured Query Language
# SQL – DML

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



## Database Management System (DBMS)






## Declarative Query

**SELECT** Title, TeamName

**FROM** SUPERBOWL

**WHERE** Rank = 1;

| Title | TeamName |
|---|---|
| Champions |  |





## SQL Select Statement

- Complete form:

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

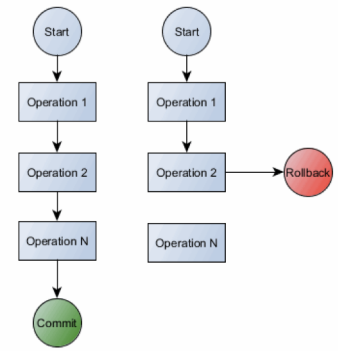



## ACID Properties

- **A**tomicity (alias failure atomicity)
  Either all the operations associated with a transaction happen or none of them happens
- **C**onsistency Preservation
  A transaction is a correct program segment. It satisfies the integrity constraints on the database at the transaction's boundaries
- **I**solation (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
- **D**urability (alias persistence / permanence)
  The effects of completed transactions become permanent surviving any subsequent failures

# SQL TRANSACTIONS

- SET TRANSACTION [READ ONLY | READ WRITE] NAME <name>;
- COMMIT ;
- ROLLBACK;
- ROLLBACK default action






# ANSI SQL2 Isolation Levels

- SET TRANSACTION READ ONLY | READ WRITE
  [ ISOLATION LEVEL READ UNCOMMITTED |
  READ COMMIT |
  REPEATABLE READ |
  SERIALIZABLE ]



# Relational Operators in SQL

- STUDENT(SID, Name, Major)

- $\Pi_{attribute\_list}(r)$:
  - $\Pi_{SID,Major}(STUDENT)$

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

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

```
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, (CURRENT_DATE – BirthDate) AS Age
  FROM LIBRARIAN
  WHERE
  (CURRENT_DATE – BirthDate) INTERVAL YEAR > 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;
```