

# CSC 443 – Database Management Systems

## Lecture 2 –Databases: The Big Picture

### Databases

- There are three different types of databases:
  - Hierarchical databases
  - Network databases
  - Relational databases
- We are particularly interested in relational databases
- In relational databases, data is stored in tables.

## What Is A Table?

- A ***table*** is a set of rows (no duplicates)
- Each row describes a different entity
- Each column states a particular fact about each entity
- Each column has an associated domain

<i>Id</i>	<i>Name</i>	<i>Address</i>	<i>Status</i>
1111	John	123 Main	fresh
2222	Mary	321 Oak	soph
1234	Bob	444 Pines	soph
9999	Joan	777 Grand	senior

- Domain of Status = {fresh, soph, junior, senior}

## What Is A Relation?

- A ***relation*** is a mathematical entity corresponding to a table
  - row ~ tuple
  - column ~ attribute
- Values in a tuple are related to each other
  - John lives at 123 Main
- Relation **R** can be thought of as predicate **R**
  - **R**(x,y,z) is true iff tuple (x,y,z) is in **R**

## What Are Operations?

- Operations on relations are precisely defined
  - Take relation(s) as argument, produce new relation as result
  - Unary (e.g., delete certain rows)
  - Binary (e.g., union, Cartesian product)
- Corresponding operations defined on tables as well
- Using mathematical properties, *equivalence* can be decided
  - Important for *query optimization*:

$$\text{op1}(\text{T1}, \text{op2}(\text{T2})) = \text{op3}(\text{op2}(\text{T1}), \text{T2})$$

## Structured Query Language: SQL

- Language for manipulating tables
- *Declarative* – Statement specifies *what* needs to be obtained, *not how* it is to be achieved (e.g., how to access data, the order of operations)
- Due to declarativity of SQL, DBMS determines evaluation strategy
  - This greatly simplifies application programs
  - *But DBMS is not infallible*: programmers should have an idea of strategies used by DBMS so they can design better tables, indices, statements, in such a way that DBMS can evaluate statements efficiently

## Structured Query Language (SQL)

```
SELECT <attribute list>  
FROM <table list >  
WHERE <condition>
```

- Language for constructing a new table from argument table(s).
  - FROM indicates source tables
  - WHERE indicates which *rows* to retain
    - It acts as a filter
  - SELECT indicates which *columns* to extract from retained rows
    - Projection
- The result is a table.

## Queries – An Example

```
SELECT Name  
FROM Student  
WHERE Id > 4999
```

<i>Id</i>	<i>Name</i>	<i>Address</i>	<i>Status</i>
1234	John	123 Main	fresh
5522	Mary	77 Pine	senior
9876	Bill	83 Oak	junior

**Student**

<i>Name</i>
Mary
Bill

**Result**

## Queries – Some Examples

```
SELECT Id, Name FROM Student
```

```
SELECT Id, Name FROM Student  
WHERE Status = 'senior'
```

```
SELECT * FROM Student  
WHERE Status = 'senior'
```

*result is a table  
with one column  
and one row*

```
SELECT COUNT(*) FROM Student  
WHERE Status = 'senior'
```

## Queries: A More Complex Example

- **Goal:** table in which each row names a senior and gives a course taken and grade
- Combines information in two tables:
  - Student: *Id, Name, Address, Status*
  - Transcript: *StudId, CrsCode, Semester, Grade*

```
SELECT Name, CrsCode, Grade  
FROM Student, Transcript  
WHERE StudId = Id AND Status = 'senior'
```

## Join

```
SELECT a1, b1  
FROM T1, T2  
WHERE a2 = b2
```

T1			T2	
<i>a1</i>	<i>a2</i>	<i>a3</i>	<i>b1</i>	<i>b2</i>
A	1	xyy	3.2	17
B	17	rst	4.8	17

```
FROM T1, T2  
yields:
```

<i>a1</i>	<i>a2</i>	<i>a3</i>	<i>b1</i>	<i>b2</i>
A	1	xyy	3.2	17
A	1	xyy	4.8	17
B	17	rst	3.2	17
B	17	rst	4.8	17

```
WHERE a2 = b2  
yields:
```

B	17	rst	3.2	17
B	17	rst	4.8	17

```
SELECT a1, b1  
yields result:
```

B	3.2
B	4.8

## Modifying Tables

```
UPDATE Student  
SET Status = 'soph'  
WHERE Id = 11111111
```

```
INSERT INTO Student (Id, Name, Address, Status)  
VALUES (999999999, 'Bill', '432 Pine', 'senior')
```

```
DELETE FROM Student  
WHERE Id = 11111111
```

## Creating Tables

```
CREATE TABLE Student (  
  Id INTEGER,  
  Name CHAR(20),  
  Address CHAR(50),  
  Status CHAR(10),  
  PRIMARY KEY (Id) )
```

*Constraint:  
explained later*

## What are Transactions?

- Many enterprises use databases to store information about their state
  - *E.g.*, balances of all depositors
- The occurrence of a real-world event that changes the enterprise state requires the execution of a program that changes the database state in a corresponding way
  - *E.g.*, balance must be updated when you deposit
- A *transaction* is a program that accesses the database in response to real-world events

## Transactions

- Transactions are not just ordinary programs
  - Additional requirements are placed on transactions (and particularly their execution environment) that go beyond the requirements placed on ordinary programs.
    - Atomicity
    - Consistency
    - Isolation
    - Durability
- } *ACID properties*

## Integrity Constraints

- Rules of the enterprise generally limit the occurrence of certain real-world events.
  - Student cannot register for a course if current number of registrants = maximum allowed
- Correspondingly, allowable database states are restricted.
  - $cur\_reg \leq max\_reg$
- These limitations are expressed as *integrity constraints*, which are assertions that must be satisfied by the database state.

## Atomicity

- A real-world event either happens or does not happen.
  - Student either registers or does not register.
- Similarly, the system must ensure that either the transaction runs to completion (*commits*) or, if it does not complete, it has no effect at all (*aborts*).
  - This is not true of ordinary programs. A hardware or software failure could leave files partially updated.

## Consistency

- Transaction designer must ensure that
  - IF the database is in a state that satisfies all integrity constraints when execution of a transaction is started
  - THEN when the transaction completes:
    - All integrity constraints are once again satisfied (constraints can be violated in intermediate states)
    - New database state satisfies specifications of transaction

## Isolation

- Deals with the execution of multiple transactions concurrently.
- If the initial database state is consistent and accurately reflects the real-world state, then the *serial* (one after another) execution of a set of consistent transactions preserves consistency.
- But serial execution is *inadequate* from a performance perspective.

## Durability

- The system must ensure that once a transaction commits its effect on the database state is not lost in spite of subsequent failures.
  - Not true of ordinary systems. For example, a media failure after a program terminates could cause the file system to be restored to a state that preceded the execution of the program.



## Isolation

- The effect of concurrently executing a set of transactions must be the same as if they had executed serially in some order
  - The execution is thus *not* serial, but *serializable*
- Serializable execution has better performance than serial, but performance might still be inadequate. Database systems offer several isolation levels with different performance characteristics (but some guarantee correctness only for certain kinds of transactions – not in general)

## ACID Properties

- The transaction monitor is responsible for ensuring atomicity, durability, and (the requested level of) isolation.
  - Hence it provides the abstraction of failure-free, non-concurrent environment, greatly simplifying the task of the transaction designer.
- The transaction designer is responsible for ensuring the consistency of each transaction, but doesn't need to worry about concurrency and system failures.