

Concordia University  
Computer Science and Software Engineering  
COMP353: Databases  
Winter 2014-2015

[http://users.encs.concordia.ca/~comp353\\_4](http://users.encs.concordia.ca/~comp353_4)

Instructor: N. Shiri ([shiri@cse.concordia.ca](mailto:shiri@cse.concordia.ca))

Lectures: MW: 13:15 – 14:30 @ H-937

Office hour: Mondays 15:00 – 16:00 @ EV 3.411

Webpage: [www.cse.concordia.ca/~shiri](http://www.cse.concordia.ca/~shiri)

## Introduction to Databases and SQL

2

## What is a Database?

- A database is a collection of data that exists over a long period of time (*Persistent storage*)
- This collection should be logically coherent and have some inherent meaning, typically about an enterprise → it may not be a random pile of data

3

## Examples of Databases

- List of names, addresses, and phone numbers of your friends
- Information about employees, departments, salaries, managers, etc. in a **COMPANY**
- Information about students, courses, grades, professors, etc. in a **UNIVERSITY**
- Information about books, users, etc. in a **LIBRARY**

4

## Database Management System (DBMS)

- A DBMS is a complex software package developed to *store* and *manage* databases
- Note the distinction between DB, DBS, and DBMS:

Database system = Database + DBMS

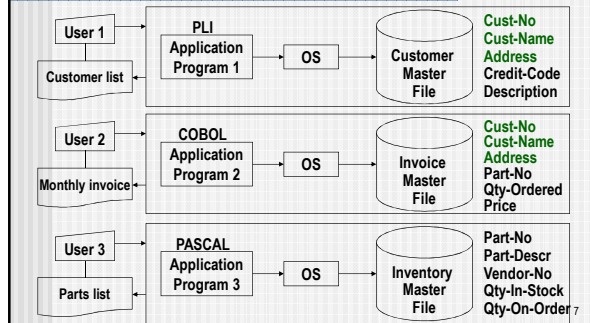
5

## What does a DBMS provide?

- Supports *convenient, efficient, and secure access and manipulation of large amounts of data*
- *(high-level) Programming interface*: Gives users the ability to create, query, and modify the data
- *Persistent storage*: Supports the storage of data over a long period of time
- *Transaction management and recovery*: Controls access to shared data from multiple, simultaneous users with properties Atomicity, Consistency, Isolation, Durability (ACID)

6

## File Processing Systems (FPS)

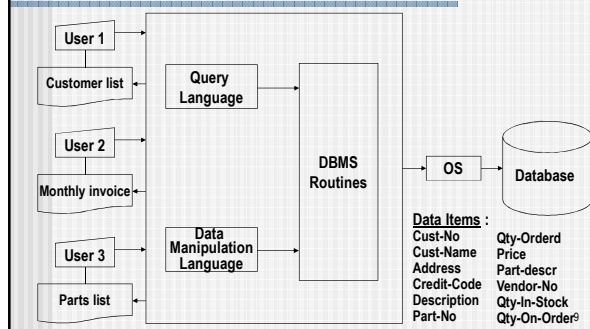


## Disadvantages of FPS

- Redundancy of data: Identical data are distributed over various files – a major source of problems
  - Waste of storage space: When the same field is stored in several files, the required storage space is needlessly high → **high storage cost**
  - Multiple updates: One field may be updated in one file but not in others → **inconsistency** and lack of data integrity and hence potential conflicting reports
  - Multiple programming languages: Dealing with several programming languages which are often not user friendly → **high system maintenance cost**

8

## Database Systems



## Advantages of Databases

- Minimize data redundancy and avoid inconsistency  
They provide:
- Concurrent access to shared data
- Centralized control over data management
- Security and authorization
- Integrity and reliability
- Data abstraction and independence

10

## Aspects of Database Studies

- Modeling and design of databases ✓
- Database programming ✓
- DBMS implementation

*The first two issues are studied in COMP353*

*The third issue is studied in COMP451*

11

## What is this course about?

- A database is a "collection of data." This data is managed by a DBMS
- Databases are essential today to support *commercial, engineering, and scientific applications*.
- They are at the core of many scientific investigations.
- Their power comes from a rich body of knowledge and technology developed over several decades
- In this course, we study fundamental concepts, techniques, and tools for *database design* and *programming*.
- In **COMP451**, we study details of *DB implementation*

12

## A quick test!

- Which one of the following is the main source of problems in file processing systems, solved by databases?
  - Waste of storage space.
  - Update anomalies, which result in lack of data integrity.
  - Data redundancy.
  - Data inconsistency.

13

## Data Modeling and Database Design

### An Overview

14

## Types of Data Models

- A Data Model is a collection of concepts, describing
  - data and relationships among data
  - data semantics and data constraints
- Entity-Relationship (ER) Model ✓
- Relational Model ✓
- Object-Oriented Model (ODL) ✓
- Logical data Model (Datalog) ✓
- Earlier "record" based Models
  - Network Model
  - Hierarchical

15

## Relational Model

- Data is organized in relations (tables)  
The user should not be concerned with the underlying storage data structure.
- Relational database schema:
  - Set of table names –  $D = \{R_1, \dots, R_n\}$
  - Set of attributes for each table –  $R_i = \{A_1, \dots, A_k\}$
- Examples of tables:
  - **Account**= {accNum, branchNam, amount, customerId}
  - **Movie**= {title, year, director, studio}

16

## Relational Model

- Most widely used model
  - Vendors: Oracle, IBM, Informix, Microsoft, Sybase, etc.
- Competitor: object-oriented model
  - ObjectStore, Postgres, etc.
- Another approach: *object-relational model*

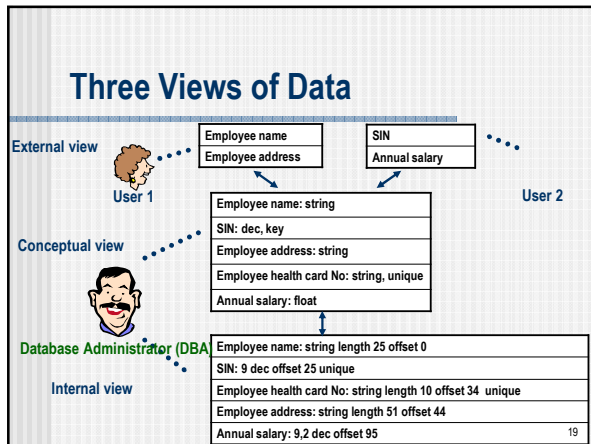
17

## Objectives of Database Systems

- A DB system should be **simple**, so that users with little skills could interact with the system **conveniently**
- It should be **complex**, so that many (complex) queries and transactions could be handled/processed **efficiently**

*But these objectives are contradictory!  
So how to achieve both?*

18



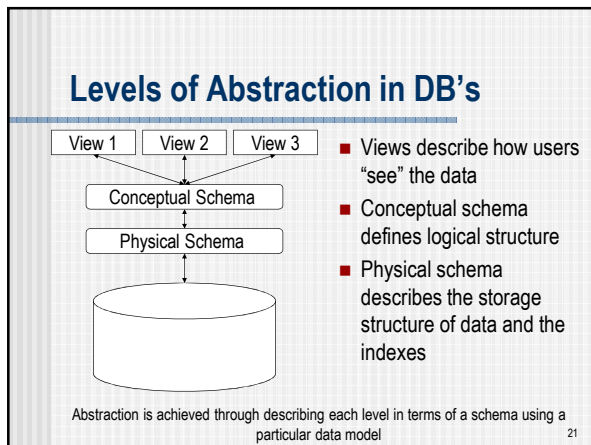
### Three Views / Levels of Data

- **Internal (physical) level**  
A block of consecutive bytes actually holding the data
- **Conceptual (logical) level**  

```

type emp = record
  SIN : integer;
  name : string;
  address : string;
  salary : real;
  healthCard : string;
end

```
- **External (logical) level**  
 View 1 : (emp.name, emp.address)  
 View 2 : (emp.SIN, emp.salary)



### Schemas at different levels of abstraction

- **View (or External Schemas):** are typically determined during requirements analysis (often defined as views over some of the concepts in the logical DB schema)
- **Conceptual (or Logical) Schema:** an outcome of a database design (a main focus in this course)
- **Physical Schema:** storage and index structures associated with relations

### Schemas and Instances

- A **database instance** is the actual content of the database at a particular point in time
- A **database schema** is a description of the structure of data, using a given data model

Students (sid, name, department, dob, address)

sid	name	department	dob	address
1112223	John Smith	CS	12-01-82	22 Pine, #1203
2223334	Ali Brown	EE	31-08-73	2000 St. Marc
3334445	Sana Kordi	CS	23-11-79	1150 Guy

### Data Independence

- **Defn:** the ability to modify definition of schema at one level without affecting the schema definition(s) at a higher level
  - Achieved through the use of three levels of data abstraction
- **Logical Data Independence**
  - Ability to modify logical schema without causing application programs to be rewritten
  - E.g., adding new fields to a record or changing the type of a field
- **Physical Data Independence**
  - Ability to modify physical schema without causing the conceptual schema or the applications to be modified, i.e., the possibility of having separate schemas at the physical and conceptual levels
  - E.g., changing a file structure from sequential to direct access

## A quick test!

- Which one of the following statements is **NOT** correct?
  - A data model is a collection of concepts for describing data and their relationships.
  - A data model is a language for describing the data semantics and the constraints.
  - A data model is a language for expressing queries and transactions in a database.
  - A data model is an abstract representation of the structure of the data.

25

## DBMS Implementation

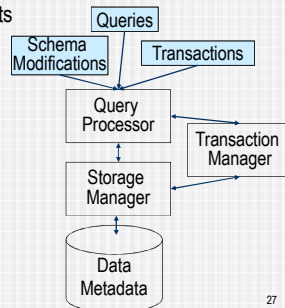
### Overview

26

## Architecture of a DBMS

- There are 3 types of inputs to DBMS:

- Queries
- Transactions, i.e., data Modifications
- Schema Creations/Modifications

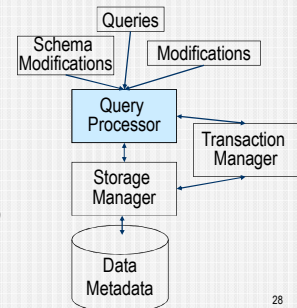


27

## Architecture of a DBMS

- The **query processor** handles:

- Queries
  - Modifications (of both data and schema)
- The job of the **query processor + query optimizer (QO)** is
    - To find the "best" plan to process the query
    - To issue commands to storage/buffer manager

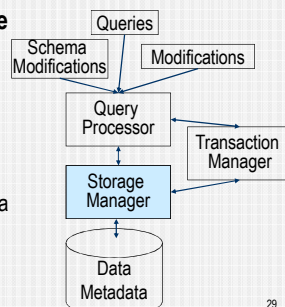


28

## Architecture of a DBMS

- The job of the **storage manager** is

- To obtain information requested **from** the data storage
- To modify the information **to** the data storage when requested.

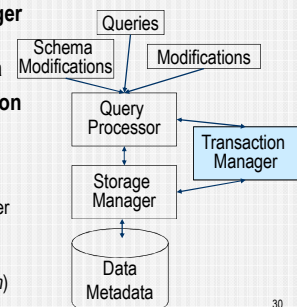


29

## Architecture of a DBMS

- The **transaction manager** is responsible for the **consistency** of the data

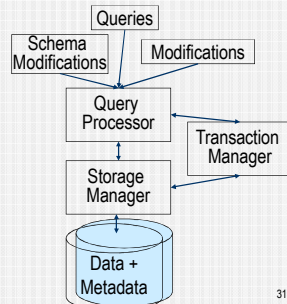
- The job of the **transaction manager** is to ensure:
  - several queries running simultaneously do not "interfere" with each other
  - Integrity of the data data even if there is a power failure (*Recovery system*)



30

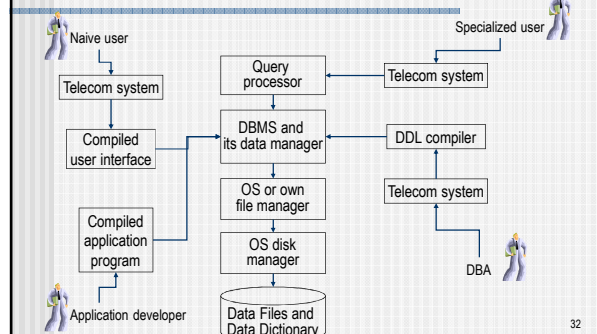
## Architecture of a DBMS

- A representation of data and other relevant information on disk
- It contains:
  - Data
  - Metadata



31

## Users of a Database System



32

## Database Programming Overview

33

## Database Languages

- A Database Management System (DBMS) provides two types of languages, which may also be viewed as components of the DBMS language:
  - Data Definition Language (DDL)
    - Language (notation) for *defining* a database schema
    - It includes syntax for declaring tables, indexes, views, constraints, etc.)
  - Data Manipulation Language (DML)
    - Language for *accessing* and *manipulating* the data (organized/stored according to the appropriate data model)

34

## Query Languages

- Commercial:
  - SQL ✓
- Theoretical:
  - Relational Algebra ✓
  - Relational Calculus
  - Datalog ✓

35

## SQL

- Developed originally at IBM in 1976
  - First standard: SQL-86
  - Second standard: SQL-92
  - Latest standard: SQL-99, or SQL3, well over 1,000 pages of document
- De-facto standard of the relational database world; replaced all other DB languages
- The SQL query language components:
  - DDL
  - DML

36

## Simple SQL Queries

- A SQL query has a form:  

```
SELECT ...  
FROM ...  
WHERE ...;
```
- The **SELECT** clause indicates which attributes should appear in the output.
- The **FROM** gives the relation(s) the query refers to
- The **WHERE** clause is a Boolean expression indicating which tuples are of interest.
- The query result is a *bag*
- Note that the query result is **unnamed**.

37

## Example SQL Query

- Relation schema:  
**Course** (courseNumber, name, noOfCredits)
- Query:  
Find all the courses stored in the database
- Query in SQL:  

```
SELECT *  
FROM Course;
```
- Note:  
“\*” means **all** attributes in the **relation(s)** involved.

38

## Example SQL Query

- Relation schema:  
**Movie** (title, year, length, filmType)
- Query:  
Find the titles of all movies stored in the database
- Query in SQL:  

```
SELECT title  
FROM Movie;
```

39

## Example SQL Query

- Relation schema:  
**Student** (ID, firstName, lastName, address, GPA)
- Query:  
Find the ID of every student whose GPA is > 3
- Query in SQL:  

```
SELECT ID  
FROM Student  
WHERE GPA > 3;
```

40

## Example SQL Query

- Relation schema:  
**Student** (ID, firstName, lastName, address, GPA)
- Query:  
Find the ID and last name of every student with first name 'John', who has a GPA > 3
- Query in SQL:  

```
SELECT ID, lastName  
FROM Student  
WHERE firstName = 'John' AND GPA > 3;
```

41

## WHERE clause

- The expressions that may follow **WHERE** are conditions
  - Standard comparison operators  $\Theta$  includes { =, <, >, <=, >= }
  - The values that may be compared include constants and attributes of the relation(s) mentioned in **FROM** clause
    - Simple expression
      - A op Value
      - A op B
      - Where A, B are attributes and op is a comparison operator
    - We may also apply the usual arithmetic operators, +, -, \*, /, etc. to numeric values before comparing them
      - (year - 1930) \* (year - 1930) < 100
    - The result of a comparison is a Boolean value **TRUE** or **FALSE**
    - Boolean expressions can be combined by the logical operators **AND**, **OR**, and **NOT**

42

## Example SQL Query

- Relation schema:  
Movie (title, year, length, filmType)
- Query:  
Find the titles of all color movies produced in 1990
- Query in SQL:  
SELECT title  
FROM Movie  
WHERE filmType = 'color' AND year = 1990;

43

## Example SQL Query

- Relation schema:  
Movie (title, year, length, filmType)
- Query:  
Find the titles of color movies that are either made after 1970 or are less than 90 minutes long
- Query in SQL:  
SELECT title  
FROM Movie  
WHERE (year > 1970 OR length < 90) AND filmType = 'color';
- Note the precedence rules, when parentheses are absent:  
AND takes precedence over OR, and  
NOT takes precedence over AND and OR

44

## Products and Joins

- SQL has a simple way to “couple” relations in one query
  - How? By listing the relevant relation(s) in the FROM clause
- All the relations in the FROM clause are coupled through Cartesian product (shown as  $\times$  in algebra notation)

45

## Cartesian Product

- From Set Theory:
  - The Cartesian Product of two sets  $R$  and  $S$  is the set of all pairs  $(a, b)$  such that:  $a \in R$  and  $b \in S$ .
  - Denoted as  $R \times S$
  - Note:
    - In general,  $R \times S \neq S \times R$

46

## A quick test!

- Let  $R(A_1, \dots, A_n)$  be a relation schema and  $r$  be any instance of  $R$ . Suppose  $r$  has  $m$  tuples. Which of the following is the number of ways in which  $r$  may be represented in the relational model?
  - $m * n$
  - $2^m$
  - $m! * n!$
  - $2^n$

47

## Example

Instance R:

A	B
1	2
3	4

Instance S:

B	C	D
2	5	6
4	7	8
9	10	11

$R \times S$ :

A	R.B	S.B	C	D
1	2	2	5	6
1	2	4	7	8
1	2	9	10	11
3	4	2	5	6
3	4	4	7	8
3	4	9	10	11

48

## Example

Instance of Student:

ID	firstName	lastName	GPA	Address
111	Joe	Smith	4.0	45 Pine av.
222	Sue	Brown	3.1	71 Main st.
333	Ann	Johns	3.7	39 Bay st.

Instance of Course:

courseNumber	name	noOfCredits
Comp352	Data structures	3
Comp353	Databases	4

SELECT \* FROM Student, Course;

ID	firstName	lastName	GPA	Address	courseNumber	name	noOfCredits
111	Joe	Smith	4.0	45 Pine av.	Comp352	Data structures	3
111	Joe	Smith	4.0	45 Pine av.	Comp353	Databases	4
222	Sue	Brown	3.1	71 Main st.	Comp352	Data structures	3
222	Sue	Brown	3.1	71 Main st.	Comp353	Databases	4
333	Ann	Johns	3.7	39 Bay st.	Comp352	Data structures	3
333	Ann	Johns	3.7	39 Bay st.	Comp353	Databases	4

9

## Example

Instance of Student:

ID	firstName	lastName	GPA	Address
111	Joe	Smith	4.0	45 Pine av.
222	Sue	Brown	3.1	71 Main st.
333	Ann	Johns	3.7	39 Bay st.

Instance of Course:

courseNumber	name	noOfCredits
Comp352	Data structures	3
Comp353	Databases	4

SELECT ID, courseNumber  
FROM Student, Course;

ID	courseNumber
111	Comp352
111	Comp353
222	Comp352
222	Comp353
333	Comp352
333	Comp353

50

## Example

- Relation schemas:
  - Student (ID, firstName, lastName, address, GPA)
  - Ugrad (ID, major)
- Query:
  - Find all information available about every undergraduate student
- We can try to compute the Cartesian product (x)
  - SELECT \* FROM Student, Ugrad;

51

## Example

Instance of Student:

ID	firstName	lastName	GPA	Address
111	Joe	Smith	4.0	45 Pine av.
222	Sue	Brown	3.1	71 Main st.
333	Ann	Johns	3.7	39 Bay st.

Instance of Ugrad:

ID	major
111	CS
333	EE

SELECT \* FROM Student, Ugrad;

ID	firstName	lastName	GPA	Address	ID	major
111	Joe	Smith	4.0	45 Pine av.	111	CS
111	Joe	Smith	4.0	45 Pine av.	333	EE
222	Sue	Brown	3.1	71 Main st.	111	CS
222	Sue	Brown	3.1	71 Main st.	333	EE
333	Ann	Johns	3.7	39 Bay st.	111	CS
333	Ann	Johns	3.7	39 Bay st.	333	EE

Which tuples should  
be in the query result  
and which shouldn't?

52

## Example

Instance of Student:

ID	firstName	lastName	GPA	Address
111	Joe	Smith	4.0	45 Pine av.
222	Sue	Brown	3.1	71 Main st.
333	Ann	Johns	3.7	39 Bay st.

Instance of Ugrad:

ID	major
111	CS
333	EE

SELECT \*  
FROM Student, Ugrad  
WHERE Student.ID = Ugrad.ID;

ID	firstName	lastName	GPA	Address	ID	major
111	Joe	Smith	4.0	45 Pine av.	111	CS
333	Ann	Johns	3.7	39 Bay st.	333	EE

53

## Join in SQL

- The above query is an example of **Join** operation
- There are different kinds of joins, which we will study!
- To join relations  $R_1, \dots, R_n$  in SQL:
  - List all these relations in the **FROM** clause
  - Express the conditions in the **WHERE** clause in order to get the "desired" join

54

## Joining Relations

- Relation schemas:
  - Movie (title, year, length, filmType)
  - Owns (title, year, studioName)
- Query: Find title, length, and studio name of every movie
- Query in SQL:

```
SELECT Movie.title, Movie.length, Owns.studioName
FROM Movie, Owns
WHERE Movie.title = Owns.title AND Movie.year = Owns.year;
```

Question: Is Owns in Owns.studioName necessary?

55

## Joining Relations

- Relation schemas:
  - Movie (title, year, length, filmType)
  - Owns (title, year, studioName)
- Query:  
Find the title and length of every movie produced by Disney studio.
- Query in SQL:

```
SELECT Movie.title, length
FROM Movie, Owns
WHERE Movie.title = Owns.title AND
      Movie.year = Owns.year AND studioName = 'Disney';
```

56

## Joining Relations

- Relation schemas:
  - Movie (title, year, length, filmType)
  - Owns (title, year, studioName)
  - StarsIn (title, year, starName)
- Query:  
Find the title and length of each movie with JR, produced by Disney
- Query in SQL:

```
SELECT Movie.title, Movie.length
FROM Movie, Owns, StarsIn
WHERE Movie.title = Owns.title AND Movie.year = Owns.year
      AND Movie.title = StarsIn.title AND Movie.year = StarsIn.year
      AND studioName = 'Disney' AND starName = 'JR';
```

## Example

Movie				Owns		
title	year	length	filmType	title	year	studioName
T1	1990	124	color	T1	1990	Disney
T2	1991	144	color	T2	1991	MGM

StarsIn				
title	year	starName	title	length
T1	1990	JR	T1	124
T2	1991	JR		

```
SELECT Movie.title, Movie.length
FROM Movie, Owns, StarsIn
WHERE Movie.title = Owns.title AND Movie.year =
      Owns.year AND Movie.title = StarsIn.title AND
      Movie.year = StarsIn.year AND studioName = 'Disney'
      AND starName = 'JR';
```

## Aggregation in SQL

- SQL provides five operators that apply to a column of a relation and produce “some kind of summary”
- These operators are called **aggregations**
- These operators are used by applying them to a scalar-valued expression, typically a column name, in a SELECT clause

59

## Aggregation Operators

- **SUM**
  - the sum of values in the column
- **AVG**
  - the average of values in the column
- **MIN**
  - the least value in the column
- **MAX**
  - the greatest value in the column
- **COUNT**
  - the number of values in the column, including the duplicates, unless the keyword **DISTINCT** is used explicitly

60

## Example

- Relation schema:  
**Exec**(name, address, cert#, netWorth)
- Query:  
Find the average net worth of all movie executives
- Query in SQL:  
**SELECT AVG**(netWorth)  
**FROM** Exec;
  - The sum of "all" values in the column **netWorth** divided by the number of these values
  - In general, if a tuple appears **n** times in a relation, it will be counted **n** times when computing the average

61

## Example

- Relation schema:  
**Exec** (name, address, cert#, netWorth)
- Query:  
How many tuples are there in the Exec relation?
- Query in SQL:  
**SELECT COUNT**(\*)  
**FROM** Exec;
  - The use of \* as a parameter is unique to **COUNT**;  
i.e., using \* does not make sense for other aggregation operations

62

## Example

- Relation schema:  
**Exec** (name, address, cert#, netWorth)
- Query:  
How many different names are there in the Exec relation?
- Query in SQL:  
**SELECT COUNT** (DISTINCT name)  
**FROM** Exec;
  - In query processing time, the system first eliminates the duplicates from the column **name**, and then counts the # of existing values

63

## Aggregation -- Grouping

- Often we need to consider the tuples in an SQL query in *groups*, with regard to the value of some other column(s)
- Example: suppose we want to compute:  
Total length in minutes of movies produced by each studio:  
**Movie**(title, year, length, filmType, studioName, producerC#)
- We must group the tuples in the Movie relation according to their studio, and get the sum of the *length* values within each group; the result would be something like:

studio	SUM(length)
Disney	12345
MGM	54321
...	...

64

## Aggregation - Grouping

- Relation schema:  
**Movie**(title, year, length, filmType, studioName, producerC#)
- Query: What is the total length in minutes produced by each studio?
- Query in SQL:  
**SELECT** studioName, **SUM**(length)  
**FROM** Movie  
**GROUP BY** studioName;
  - Whatever aggregation used in the **SELECT** clause will be applied only within groups
  - Only those attributes mentioned in the **GROUP BY** clause may appear unaggregated in the **SELECT** clause
  - Can we use **GROUP BY** without using aggregation?

65

## Aggregation -- Grouping

- Relation schema:  
**Movie**(title, year, length, filmType, studioName, producerC#)  
**Exec**(name, address, cert#, netWorth)
- Query:  
For each producer (name), list the total length of the films produced
- Query in SQL:  
**SELECT** Exec.name, **SUM**(Movie.length)  
**FROM** Exec, Movie  
**WHERE** Movie.producerC# = Exec.cert#  
**GROUP BY** Exec.name;

66

## A rule about null values!

- Nulls counted in grouping but ignored in aggregating.  
Ex. Consider the instance below of R(A,B) with the tuples.  
Which one of the following tuples is *not* in the result of the query below?

```
Select A, Sum(B)
From R
Group By A;
```

- (null, null)
- (2,4)
- (1,null)
- (null,1)

A	B
null	1
2	1
null	null
3	2
2	3
1	null

67

## Another test!

- Consider again the same instance of R(A,B) containing the tuples: (null,1), (2,1), (null, null), (3,2), (2,3), and (1,null). Which one of the following tuples will be in the result of the query below?

```
Select A, Sum(B)
From R
Where B <> 2
Group By A;
```

- (null, 0)
- (1,null)
- (2,3)
- (2,4)

68

## Answer!

- Consider an instance of R(A,B) with the tuples (null,1), (2,1), (null, null), (3,2), (2,3), and (1,null). Which one of the following tuples will be in the result of the query below?

```
Select A, Sum(B)
From R
Where B <> 2
Group By A;
```

- |           |        |
|-----------|--------|
| (null, 0) | null 1 |
| (1,null)  | 2 1    |
| (2,3)     | 2 3    |
| ✓ (2,4)   |        |

69

## Aggregation – HAVING clause

- We might be interested in not all but some groups of tuples that satisfy certain conditions
- We can follow a **GROUP BY** clause with a **HAVING** clause
- HAVING** is followed by some conditions about the group
- We can *not* use a **HAVING** clause without **GROUP BY**

70

## Aggregation – HAVING clause

- Relation schema:
  - Movie** (title, year, length, filmType, studioName, producerC#)
  - Exec**(name, address, cert#, netWorth)
- Query:
  - For those producers who made at least one film prior to 1930, list the total length of the films produced
- Query in SQL:
 

```
SELECT Exec.name, SUM(Movie.length)
FROM Exec, Movie
WHERE producerC# = cert#
GROUP BY Exec.name
HAVING MIN(Movie.year) < 1930;
```

71

## Aggregation – HAVING clause

- This query chooses the group based on the **property of each group**

```
SELECT Exec.name, SUM(Movie.length)
FROM Exec, Movie
WHERE producerC# = cert#
GROUP BY Exec.name
HAVING MIN(Movie.year) < 1930;
```
- But this one chooses the movies based on the **property of each movie**

```
SELECT Exec.name, SUM(Movie.length)
FROM Exec, Movie
WHERE producerC# = cert# AND Movie.year < 1930
GROUP BY Exec.name;
```

*Note the difference!*

72

## Order By

- The SQL statements/queries we looked at so far return an **unordered relation/bag**. What if we want the result displayed in a certain order?

**Movie** (title, year, length, filmType, studioName, producerC#)

```
SELECT Exec.name, SUM(Movie.length)
FROM Exec, Movie
WHERE producerC# = cert#
GROUP BY Exec.name
HAVING MIN(Movie.year) < 1930
ORDER BY Exec.name ASC;
```

In general:  
**ORDER BY A ASC, B DESC, C ASC;**

73

## Database Modifications

### SQL & Database Modifications?

- We now look at SQL statements that do not return tuples, but rather **change the state (content) of the database**
- There are three types of such statements/transactions:
  - **Insert** tuples into a relation
  - **Delete** certain tuples from a relation
  - **Update** values of certain attributes of certain existing tuplesThese types of operations that modify the database content are referred to as **transactions**

74

## Insertion

- The insertion statement consists of:
  - The keyword **INSERT INTO**
  - The name of a relation **R**
  - A parenthesized list of attributes of the relation **R**
  - The keyword **VALUES**
  - A tuple expression, that is, a parenthesized list of concrete values, one for each attribute in the attribute list
- The form of an insert statement:  
**INSERT INTO R(A<sub>1</sub>, ..., A<sub>n</sub>) VALUES (v<sub>1</sub>, ..., v<sub>n</sub>);**
  - This command adds tuple (v<sub>1</sub>, ..., v<sub>n</sub>) to table **R**, where v<sub>i</sub> is the value of attribute A<sub>i</sub>, for i = 1, ..., n

75

## Insertion

- Relation schema:  
**StarsIn** (title, year, starName)
- Update the database:  
Add "Sydney Greenstreet" to the list of stars of *The Maltese Falcon*
- In SQL:  
**INSERT INTO StarsIn (title, year, starName)**  
**VALUES('The Maltese Falcon', 1942, 'Sydney Greenstreet');**  
*Another formulation of this query:*  
**INSERT INTO StarsIn**  
**VALUES('The Maltese Falcon', 1942, 'Sydney Greenstreet');**

76

## Insertion

- The previous insertion statement was "simple" in that it added just **one** tuple into a relation
- Instead of using *explicit* values for one tuple, we can request a **set** of tuples to be inserted. For this we define, in a *subquery*, the set of tuples from an existing relation
- This subquery replaces the keyword **VALUES** and the tuple expression in the **INSERT** statement

77

## Insertion

- Database schema:  
**Studio** (name, address, presC#)  
**Movie** (title, year, length, filmType, **studioName**, producerC#)
- Update the database:  
Add to **Studio**, all **studio names** mentioned in the **Movie** relation
- Note: If the list of attributes in an insert statement does not include all the attributes of the relation, the tuple created will have the **default** value for each missing attribute
- Since there is no way to determine an **address** or a **presC#** for a studio tuple, **NULL** will be used for these attributes.

78

## Insertion

- Database schema:  
**Studio**(name, address, presC#)  
**Movie**(title, year, length, filmType, **studioName**, producerC#)
- Update the database:  
Add to **Studio**, all **studio names** mentioned in the **Movie** relation
- In SQL:  

```
INSERT INTO Studio(name)
SELECT DISTINCT studioName
FROM Movie
WHERE studioName NOT IN (SELECT name
                          FROM Studio);
```

79

## Deletion

- A delete statement consists of :
  - The keyword **DELETE FROM**
  - The name of a relation **R**
  - The keyword **WHERE**
  - A condition
- The syntax of the delete statement:  
**DELETE FROM R WHERE <condition>;**
  - The effect of executing this statement is that "every tuple" in relation **R** satisfying the condition will be deleted from **R**
  - Note: unlike the INSERT, we MAY need a WHERE clause here

80

## Deletion

- Relation schema:  
**StarsIn**(title, year, starName)
- Update:  
Delete the tuple that says:  
*Sydney Greenstreet was a star in The Maltese Falcon*
- In SQL:  

```
DELETE FROM StarsIn
WHERE title = 'The Maltese Falcon' AND
      starName = 'Sydney Greenstreet';
```

81

## Deletion

- Relation schema:  
**Exec**(name, address, cert#, netWorth)
- Update:  
*Delete every movie executive whose net worth is < \$10,000,000*
- In SQL:  

```
DELETE FROM Exec
WHERE netWorth < 10,000,000;
```

Anything wrong here?!

82

## Deletion

- Relation schema:  
**Studio**(name, address, presC#)  
**Movie**(title, year, length, filmType, **studioName**, producerC#)
- Update:  
Delete from **Studio**, those studios not mentioned in **Movie**  
(i.e., we don't want to have non-productive studios!!)
- In SQL:  

```
DELETE FROM Studio
WHERE name NOT IN (SELECT StudioName
                  FROM Movie);
```

83

## Update

- Update statement consists of:
  - The keyword **UPDATE**
  - The name of a relation **R**
  - The keyword **SET**
  - A list of formulas, each of which will assign a value to an attribute of **R**
  - The keyword **WHERE**
  - A condition
- The syntax of the update statement:  
**UPDATE R SET <new-value assignments> WHERE <condition>;**

84

## Update

- Database schema:  
`Studio(name, address, presC#)`  
`Exec(name, address, cert#, netWorth)`
- Update:  
Modify table `Exec` by attaching the title 'Pres.' in front of the name of every movie executive who is also the president of some studio
- In SQL:  
`UPDATE Exec`  
`SET name = 'Pres.' || name` ← this line performs the update  
`WHERE cert# IN (SELECT presC#`  
`FROM Studio);`

85

## Defining Database Schema

- SQL includes two types of statements:
  - DML
  - DDL
- So far we looked at the DML part to specify or modify the relation/database instances.
- The DDL part allows us to define or modify the relation/database schemas.

86

## Defining Database Schema

- To create a table in SQL:
  - `CREATE TABLE name` (list of elements);
    - Principal elements are *attributes* and their *types*, but declarations of *key* and *constraints* may also appear
  - Example:  
`CREATE TABLE Star (`  
`name CHAR(30),`  
`address VARCHAR(255),`  
`gender CHAR(1),`  
`birthdate DATE`  
`);`

87

## Defining Database Schema

- To delete a table from the database:
  - `DROP TABLE name`;
- Example:  
`DROP TABLE Star`;

88

## Data types

- `INT` or `INTEGER`
- `REAL` or `FLOAT`
- `DECIMAL(n, d)` -- `NUMERIC(n, d)`
  - `DECIMAL(6, 2)`, e.g., 0123.45
- `CHAR(n)`/`BIT(B)` fixed length character/bit string
  - Unused part is padded with the "pad character", denoted as `_`
- `VARCHAR(n)` / `BIT VARYING(n)` variable-length strings up to `n` characters
- Oracle also uses `VARCHAR2(n)`, which is truly varying length;  
Since `VARCHAR` uses fixed array with `end-marker`, it is not followed any longer in Oracle.

89

## Data types (cont'd)

- **SQL2 Syntax for:**
  - **Time:** 'hh:mm:ss[.ss...]'
  - **Date:** 'yyy-mm-dd' (m=0 or 1)
- Example:  
`CREATE TABLE Days(d DATE);`  
`INSERT INTO Days VALUES('2012-12-23');`
  - ❖ **Note 1:** In Oracle, the default format of date is 'dd-mon-yy', e.g.,  
`INSERT INTO Days VALUES('12-jan-18');`
  - ❖ **Note 2:** Oracle function `to_date` converts a specified format into default, e.g.,  
`INSERT INTO Days VALUES (to_date('2002-08-08', 'yyyy-mm-dd'));`

90

## Altering Relation Schemas

- **Adding Columns**
    - Add an attribute to an existing relation R:  
`ALTER TABLE R ADD <column declaration>;`
    - **Example: Add attribute phone to table Star**
      - `ALTER TABLE Star ADD phone CHAR(16);`
  - **Removing Columns**
    - Remove an attribute from a relation R using DROP:
      - `ALTER TABLE R DROP COLUMN <column_name>;`
    - **Example: Remove column phone from Star**
      - `ALTER TABLE Star DROP COLUMN phone;`
- Note: Can't drop a column, if it is the only column**

91

## Attribute Properties

- We can assert that the value of an attribute A to be:
  - **NOT NULL**
    - Then every tuple must have a "real" value (not null) for this attribute
  - **DEFAULT value**
    - Null is the default value for every attribute
    - However, we can consider/define any value we wish as the default for a column, when we create a table.

92

## Attribute Properties

```
CREATE TABLE Star (  
  name CHAR(30),  
  address VARCHAR(255),  
  gender CHAR(1) DEFAULT '?',  
  birthdate DATE NOT NULL);
```

- Example: Add an attribute with a default value:
  - `ALTER TABLE Star ADD phone CHAR(16) DEFAULT 'unlisted';`
- `INSERT INTO Star(name, birthdate) VALUES ('Sally', '0000-00-00')`

name	address	gender	birthdate	phone
Sally	NULL	?	0000-00-00	unlisted
- `INSERT INTO Star(name, phone) VALUES ('Sally', '333-2255');`
  - this insertion op. fails since the value for `birthdate` is not given, since it was disallowed for this att. to be NULL as the default

93