Database Design

Structured Query Language

PWSP Workshop

February 26-27

Sanjay Goel

Structured Query Language

Definition

- The Structured Query Language (SQL) comprises one of the fundamental building blocks of modern database architecture.
- SQL defines the methods used to create and manipulate relational databases on all major platforms.
- SQL commands can be divided into two main sublanguages.
  - The Data Definition Language (DDL) contains the commands used to create and destroy databases and database objects.
  - Data Manipulation Language is used to insert, retrieve and modify the data contained in a database

Data Definition Language

Structured Query Language

Learning Objectives

- The Data Definition Language (DDL) is used to create and destroy databases and database objects.
- The primary set of commands are
  - Create
  - Insert
  - Delete
  - Drop
  - Truncate
  - Alter

Data Definition Language

CREATE Command

- Create command is universal command to create objects in the database
- It can be used to create
  - Database
  - Tables
  - Views
  - Indexes

Data Definition Language

Creating a Database

To create a new database

Syntax: Create database database_name

- There are numerous arguments that go along with this command but are database specific
- Only some databases require database to be created and space to be allocated prior to creation of tables.
- Some databases provide graphical user interfaces to create databases and allocate space.
  - Access only allows database to be created using User Interface

Data Definition Language

Creating a Table

- **Syntax**

```
Create table table_name
(Column_name        datatype[(size)],
 Column_name        datatype[(size)],
)
```

- **Example**

```
Create table books
(ISBN               char(20),
Title               char(50),
AuthorID            Integer,
Price               float)
```

- Creates a table with four columns

Data Definition Language

Data Types

- Following broad categories of data types exist in most databases:
  - String Data
  - Numeric Data
  - Temporal Data
  - Large Objects

Data Definition Language

String Data: Fixed Length

- Fixed Length strings occupy the same length of space in memory no matter how much data is stored in them.
- **Syntax:**

  char(n) where n is the length of the String

  e.g. name char(50)
- If the variable stored for name is 'Sanjay' the extra 43 fields are padded with blanks

Data Definition Language

String Data: Variable Length

- Variable Length string is specified with maximum length of characters possible in the string, however, the allocation is sized to the size of the data stored in memory.
- **Syntax:** Varchar(n)
  - n is the maximum length of data possible for the type
- There may be a restriction in the maximum length of the data that you can specify in the declaration which will vary according to the database.
- All character data has to be enclosed in single quotes during specification.

Data Definition Language

Numeric Data Types

- Store all the data related to purely numeric data.
- Some numeric data may also be stored as a character field e.g. zip codes
- **Common Numeric Types**:
  - Decimal — Floating point number
  - Float — Floating point number
  - Integer(size) — Integer of specified length
  - Money — A number which contains exactly two digits after the decimal point
  - Number — A standard number field that can hold a floating point data

Note: Different databases name their numeric fields differently and may not support all numeric types. They may also support additional numeric types.

Data Definition Language

Temporal Data Types

- **These represent the dates and time:**
- Three basic types are supported:
  - Dates
  - Times
  - Date-Time Combinations

Data Definition Language

Large Data Objects

- These are used for storing data objects like files and images:
- **There are two types:**
  - Character Large Objects (clobs)
  - Binary Large Objects (blobs)

Data Definition Language

Specifying Keys

- Unique keyword is used to specify keys.
  - This ensures that duplicate rows are not created in the database.
- Both Primary keys and Candidate Keys can be specified in the database.
- Once a set of columns has been declared unique any data entered that duplicates the data in these columns is rejected.
- **Specifying a single column as unique**:
- Example

```
Create Table Studios
(studio_id          Number,
name                char(20),
city                varchar(50),
state               char(2),
Unique (name))
```

- Here the name column has been declared as a candidate key

Data Definition Language

Specifying Keys

- Specifying multiple columns as unique:
- **Example:**

```
Create Table Studios
(studio_id          Number,
name                char(20),
city          varchar(50),
state         char(2),
Unique (name),
Unique(city, state))
```

- Here both name & city/state combination are declared as candidate keys

Data Definition Language

Insert Command

- Insert command allows you to add new records to the Table
- **Syntax**: Insert into table_name [(column_list)] values (value_list)
- **Example 1**:

  Insert into studios

  Values (1, 'Giant', 'Los Angeles', 'CA')
- **Example 2**:

  Insert into studios

  (studio_city, studio_state, studio_name, studio_id)

  Values ('Burbank', 'CA', 'MPM', 2)
- **Note**: If the columns are not specified as in the previous example the data goes in the order specified in the table

Data Definition Language

Inserting Null Values

- There are two ways of inserting Null values
- If the field has a default value of Null, you can use an Insert statement that ignores the column where the value is to be Null.
- **Example 1**:

  Insert into studios

  Values (1, 'Giant', , 'CA')
- You can specify the column in the column list specification and assign a value of Null to the corresponding value field.
- **Example 2**:

  Insert into studios

  (studio_city, studio_state, studio_name, studio_id)

  Values ('Burbank', 'CA', NULL, 2)

Data Definition Language

Select and Insert

- Select & Insert: A select query can be used in the insert statement to get the values for the insert statement
- **Example**:

  Insert into city_state

  Select studio_city, studio_state from studios

  Note: This selects the corresponding fields from the studios table and inserts them into the city_state table.
- **Example:**

  Insert into city_state

  Select Distinct studio_city, studio_state from studios

  Note: This selects the corresponding fields from the studios table, deletes the duplicate fields and inserts them into the city_state table. Thus the final table has distinct rows

Data Definition Language

Delete

- Delete statement is used to remove records from a table of the database. The where clause in the syntax is used to restrict the rows deleted from the table otherwise all the rows from the table are deleted.
- **Syntax:** Delete From table_name [Where Condition]
- Example:

  Delete From City_State

  Where state = 'TX'
- Deletes all the rows where the state is Texas keeps all the other rows.

Data Definition Language

Update

- Update statement is used to change existing rows of the table.
- It has three parts.
  - **Update clause** specifies which table is going to be updated.
  - **Set clause** specifies columns that will be updated as well as the values that will be inserted.
  - **Where clause** is used to specify which rows will be updated.
- **Syntax**:

  Update table_name

  Set column_name1 = value1, column_name2 = value2, …..

  [Where Condition]
- **Example**:

  Update studios

  Set studio_city = 'New York', studio_state = 'NY'

  Where studio_id = 1
- **Notes**: If the condition is dropped then all rows are updated.

Data Definition Language

Update

- Truncate deletes all the rows of a table
  - Delete can also be used to delete all the rows from the table.
  - The difference is that delete performs a delete operation on each row in the table and performs integrity checks while the truncate statement simply throws away all the rows at once
  - Truncate is thus much quicker than delete
  - Since truncate does not perform integrity checks it can lead to inconsistencies on the way.
  - If there are dependencies requiring integrity checks we should use delete.
- **Syntax**: Truncate Table table_name
- **Example**:

  Truncate Table Studios

  Deletes all the rows of the table studios

Data Definition Language

Drop

- Drop used to remove elements from a database, such as tables, indexes or even users and databases.
- Drop command is used with several keywords based on the need, e.g.
  - **Drop Table**

    Syntax: Drop Table table_name

    Example: Drop Table studios
  - **Drop Index**

    Syntax: Drop Index table_name

    Example: Drop Index movie_index

Data Definition Language

Alter: Add a column

- Alter is used to make changes to the schema of the table.
  - Columns can be added
  - Constraints can be added
  - data type of the columns can be changed as long as the data in the columns conforms to the data type specified.

Add a column

- **Syntax**:

  Alter Table table_name

  Add (column datatype [Default Expression])

  [References table_name (column_name)'

  [Check condition]

- **Example**:

  Alter Table studios

  Add (revenue Number Default 0)

Data Definition Language

Alter: Add constraints

- **Add table level constraints**
- **Syntax**:

  Alter Table table_name

  Add ([Constraint constraint_name Check comparison]

  [columns References table_name (columns)]
- **Example**:

  Alter Table studios

  Add (Constraint check_state check (studio_state in ('TX', 'CA', 'WA'))

Data Definition Language

Alter: Modify Columns

- **Syntax:**

  Alter Table table_name
  Modify column [data type]
  [Default Expression]
  [References table_name (column_name)'
  [Check condition]

- **Example:**

  Alter Table People
  Modify person_union varchar(10)

- Columns can not be removed from the table using alter.
- If you want to remove columns you have to drop the table and then recreate it without the column that you want to discard

Relational Algebra

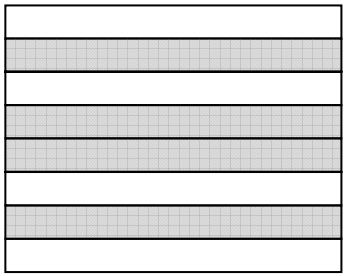
Relational Algebra

Definition

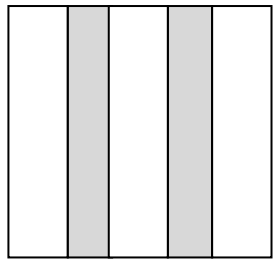
- Relational Algebra is Query Language
  - Collection of high level operators that operate on relations.
  - Theoretical, Procedural Language
  - Purpose is data manipulation
  - Method is to write expressions
  - Six Fundamental Operators
  - Other operators defined in terms of fundamental operators
- SQL can be mapped into relational algebra operations

Relational Algebra

Pictorial Representation



Select



Project

a
b
c

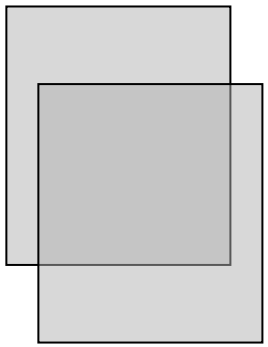
x

x
y

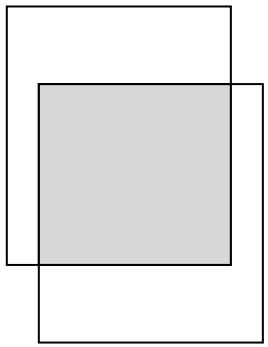
→

a	x
a	y
b	x
b	y
c	x
c	y

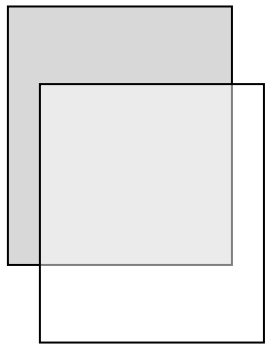
Cartesian Product



Union



Intersection



Difference

Rename

a1	b1
a2	b2
a3	b3

b1	c1
b2	c2
b3	c3

→

a1	b1	c1
a2	b2	c2
a3	b3	c3

Join

a	x
a	y
a	z
b	x
c	y

x
y

→

a

Divide

Relational Algebra

Example

Given:

1. Animal (Animal_name, food, nlegs)
2. Keeper(keeper#, keeper_name)
3. Supervision(keeper#, animal_name)

Queries:

1. What does a camel eat?
   - (PROJECT, RESTRICT)
2. What is supervised by a keeper called Morris?
   - (JOIN, RESTRICT, PROJECT)

Relational Algebra

Example

Given:

1. Book (ISBN, Price, Title)
2. Author(AuthorID, AuthorName)
3. Book/Author(AuthorID, ISBN)

Queries:

1. What is the Price of the book "War and Peace"?
   - (PROJECT, RESTRICT)
2. Who is the author of the book War and Peace?
   - (JOIN, RESTRICT, PROJECT)
3. Find all the books written by author Shakespeare?
   - (JOIN, RESTRICT, PROJECT)

Data Manipulation Language

Select Clause

Syntax

Select *<List of Columns and expressions (usually involving columns)>*

From *<List of Tables & Join Operators>*

Where *<List of Row conditions joined together by And, Or, Not>*

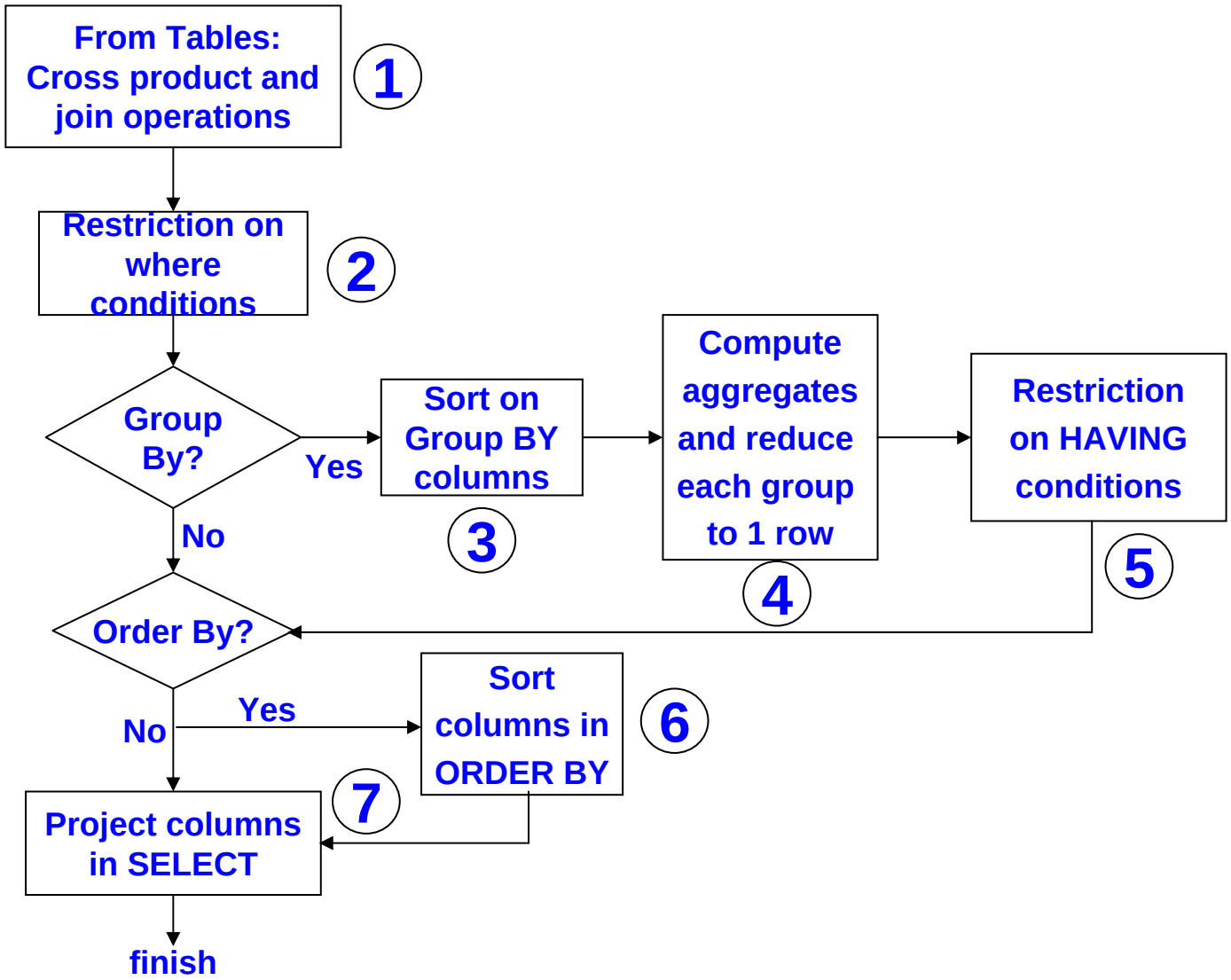
Group By *<list of grouping columns>*

Having *<list of group conditions connected by And, Or, Not >*

Order By *<list of sorting specifications>*

Select Clause

Conceptual Evaluation



Select Clause

Example

- Query:

```
Select movie_title, studio_id
From Movies
Where movie_type = 'Comedy'
```

Output:

Movie_Title	Studio_ID
Vegetable house	1
Broccoli Wars	2
Carrot Affairs	4
Chocolat	1
Cranberry House	2

- Notes:
  - Database looks in the movie_type column until it locates a comedy. When it finds comedy it retrieves the value of movie_title & studio_id
  - The where clause is optional. When not specified the columns from all the records are extracted.
  - Changing the order in the select_list changes the order in which the columns are displayed
  - Using a * for the select_list selects all the columns from the table. They are listed in the same order as in the original table.

Select Clause

Expressions in Select List

- Expressions can be used to change the values prior to printing
- **Example**:

```
Select 'Random Text' movie_title, studio_id, 2 + 2
From Movies
Where movie_type = 'Comedy'
```

Output:

RandomText	Movie_Title	Studio_ID	2+2
'Random Text'	Vegetable house	1	4
'Random Text'	Broccoli Wars	2	4
'Random Text'	Carrot Affairs	4	4
'Random Text'	Chocolat	1	4
'Random Text'	Cranberry House	2	4

Select Clause

Expressions in Select List

- **Example:**

  Select movie_title, gross, gross*1.5

  From Movies

  Output:

Movie_Title	gross	gross*1.5
Vegetable house	30	45
Broccoli Wars	20	30
Carrot Affairs	11	16.5
Chocolat	10	15
Cranberry House	50	75

Select Clause

Operators

- Arithmetic operators supported by SQL
  - () Parentheses
  - / Division
  - * Multiplication
  - - Subtraction
  - + Addition
- Associativity and Precedence:
  - Precedence is the order in which operators are evaluated
  - Associativity is the order in which operators of same precedence are evaluated
  - Multiplication and Division have the same precedence and Subtraction and Division have the same precedence.
  - Equal precedence operators are evaluated from right to left
  - Parentheses can be used to control the sequence of evaluation of various operators

Select Clause

Alias (as)

- Used to assign names to the columns when they are retrieved from the database table.
- **Syntax**:

  Select expr1 [as alias1], expr2 [as alias2] [, …]

  From table1 [, table2, …]

  [Where condition]

- **Example**:

  Select city, ((1.8 + avg_temp) + 32) AS temperature

  From Temperature

  Output

City	Temperature
London	61.7
Albany	78.4
Paris	66.2

Select Clause

Alias (as)

- A multiword Heading needs to be enclosed in double quotes
- **Example**:

```
Select city, ((1.8 + avg_temp) + 32) AS "Average Temperature"
From Temperature
```

Output:

City	Average Temperature
London	61.7
Albany	78.4
Paris	66.2

Where Clause

Basics

- Conditional statements in the select clause restrict the selection of rows in the database.
- It can be used in a variety of SQL Statements
- **Syntax**:
  - Update table Set (column = value, column = …) [Where condition]
  - Delete From table [Where condition]
  - Select list from table [Where condition]
- Condition is a Boolean expression which evaluates to true or false
- Complex expressions can be generated by using logical operators

Where Clause

Operators

- Arithmetic Operators used in the where clause
  - = equal
  - <>, != not equal
  - > Greeater Than
  - < Less Than
  - >= Greater than or equal to
  - <= Less than or equal to
- Logical operators
  - AND
  - OR
  - NOT
- For numeric operator comparison you should not use quotes around the number
- You should put single quotes around characters and strings

Where Clause

Null Values

- Null values are unknown so the regular operators can not be used for comparison
  - IS NULL is used to check if the field contains a null value or not.
  - IS NOT NULL is used to see if a field is not null
- Example

```
Select movie_title
From movies
Where gross is null

Select movie_title
From movies
Where gross is not null
```

Where Clause

Examples

- **Example**:

```
Select movie_title, studio_id , gross
From Movies
Where studio_id = 3 and gross Is Null
```

Output:

Movie_Title	Studio_ID	GROSS
Bill Durham	3	

- **Example**:

```
Select movie_title, studio_id , gross
From Movies
Where studio_id = 3 OR gross Is Null
```

Output

Movie_Title	Studio_ID	GROSS
Bill Durham	3	
Prince Kong	2	
SQL Strikes Back	3	10
The Programmer		25.5

Where Clause

Examples

- **Example:**

```
Select movie_title, studio_id , gross
From Movies
Where studio_id = 3 and NOT gross Is Null
```

Output

Movie_Title	Studio_ID	GROSS
SQL Strikes Back	3	10
The Programmer	3	25.5

- **Example:**

```
Select movie_title, studio_id, gross
From Movies
Where studio_id = 3
      or studio_id = 2
      or studio_id = 1
```

- Output

Movie_Title	Studio_ID	GROSS
SQL Strikes Back	3	10
The Programmer	3	25.5

Where Clause

IN condition

- **IN** condition checks if the values in a column are present in list list when selecting
- **Syntax:**

  Select select_list

  From table

  Where column [not] in (value_list)
- **Example (Using IN):**

  Select movie_title, studio_id

  From Movies

  Where studio_id in(2, 3)
- **Example (not Using IN)**

  Select movie_title, studio_id

  From Movies

  Where studio_id = 2

  or studio_id = 3
- NOT IN can similarly be used to select rows where values do not match

Where Clause

Between condition

- **Between** condition is used to see if the value of a column lies between specified ranges
- **Syntax:**
  - Select movie_title, budget
  - From table
  - Where column [not] between lower_value and upper_value
- **Example:**

  Select movie_title, budget

  From Movies

  Where budget between 10 and 50
- **Alternate Query:**

  Select movie_title, budget

  From Movies

  Where budget > 10 and budget < 50

Where Clause

Like

- **Like** allows a matching of patterns in the column data
- **Syntax:**
  - Select select_list
  - From table
  - Where column [not] like 'pattern' [Escape char]
- **Wildcards:**
  - - Any Single Character
  - % (or *) 0 or more characters
  - A combination of '-' and '%' can be used to mean 1 or more
- For test of fixed number of characters multiple dashes can be used
  - For example '----' will select all 3 letter words from the column
- **Example:**

```
Select movie_title
From movies
Where movie_title like 'The %'
```

Output:

```
movie_title
------------
The Code Warrier
The Linux Programmer
The Rear Windows
```

Where Clause

Escaping wild card characters

- SQL allows you to define your own escape characters if you want to include the % as a part of the search string.
- **Example:**

```
Select movie_title
From movies
Where movie_title like '%50\%%' ESCAPE \
```

- This shows that the escape character is \

Where Clause

String Comparison

- **Example**

  Select movie_title, studio_id
  From Movies
  Where movie_title = 'Independence Day'

- Output

  Movie_title Stuodio_ID

  Independence Day 1

- **Functions for where clauses**
  - Upper()
  - Lower()
  - Trim()
  - Length()

- **Example:**

  Select studio_name
  From Studios
  Where lower(studio_state) = 'ca'

Output:

Studio_name

Giant
Mpm
Metaversal Studios

Where Clause

Expressions

- Similar to the expressions in the select clause
- **Example:**

```
Select movie_title, gross, budget
From movies
Where gross > (2 * budget)
```

Output

```
Movie_Title          Gross      budget
-----------------------------------------
Prince Kong          51.5       3.25
```

Select Clause

Distinct

- Eliminates all the duplicate entries in the table resulting from the query.

Syntax:

Select [DISTINCT] select_list
From table[, table, …]
[Where expression]
[Order By expression]

Example:

Select DISTINCT studio_id, director_id
From Movies

studio_id	director_id
1	1
2	2
2	10
3	1
3	9

Select Clause

Distinct

- Eliminates all the duplicate entries in the table resulting from the query.

Syntax:

Select [DISTINCT] select_list
From table[, table, …]
[Where expression]
[Order By expression]

Example:

Select DISTINCT studio_id, director_id
From Movies

studio_id	director_id
1	1
2	2
2	10
3	1
3	9

Select Clause

Order By - Syntax

- Used to sort the results based on contents of a column
- Multiple levels of sort can be done by specifying multiple columns
- An expression can be used in Order By clause

Syntax:

Select function (column)

From table1 [, table2 …]

[Where condition]

[Order By {Column | alias | position} [ASC | DESC]]

Select Clause

Order By - Example

Query: Sort Movies by profits in Ascending order

Select MovieTitle, Gross, Budget, (Gross – Budget) as profits

From movies

Order BY profits

Movie_title	Gross	Budget	Profit
Great Escape	67.5	70	-2.5
Upside Down	54	50	4
Green Warrior	96	80	16
Blue Oranges	28	7	21

Select

Aggregate Queries

- Aggregate queries provides a more holistic view of the data by further processing the retrieved data.
- Categorizes the query results according to the contents of a column in the database
- Multiple levels of subgroups can be created by specifying multiple columns
- They can work on
  - On all the rows in a table
  - A subset of rows in a table selected using a where clause
  - Groups of selected data organized using Group By clause.

Select - Aggregate Queries

Group By (Syntax)

Syntax:

Select function(column)

From <list of tables>

Where <condition>

Group By <list of columns>

Having <condition>

Aggregate Queries

Functions

- Functions:

Sum()	Returns a sum of the column
Count()	Returns a total number of rows returned by a query
Avg()	Returns the average of a column
Min()	Returns minimum value of the column returned by query
Max()	Returns maximum value of the column returned by query

  - Count function
    - does not include columns containing null values in total
    - can be used with distinct to count the number of distinct rows

Example:

Query:
```
Select sum(budget)
From movies
Where studio_id = 3
```

Output:
```
Sum(budget)
--------------
65.1
```

Select - Aggregate Queries

Group By (Examples)

Problem 1:

Get # of movies by each director for each studio

```
Select studio_id, director_id, count(*)
From Movies
Group By director_id, studio_id
```

Problem 2:

Get # of movies by each studio ordered by studio_id

```
Select studio_id, count(*)
From Movies
Group By studio_id
Order By studio_id
```

Select - Aggregate Queries

Group By (Examples)

Problem 3: (Summation)

```
Select studio_id, Sum(budget)
    From movies
    Group  by studio_id
    Having Sum(budget) > 60
```

Problem 4: (Count)

```
Select studio_id, count(*)
    From Movies
    Group By studio_id
    Order By studio_id
```

Join Queries
Definition

- A Join Query uses data from multiple tables
  - Multiple tables are specified in the From Clause
  - A join query without any restrictions will join every row in one table with each row in the other table.
  - For two tables to be joined in a sensible manner, they need to have data in common
  - The join condition should usually specify the foreign key equivalence condition

Problem: Get names of the directors for movies listed in the movie table

Schema: Movies (movie_title, director_id, release_date)

People(person_fname, person_lname, person_id)

Query:
```
Select movie_title, person_fname, person_lname
    From Movies, People
    Where director_id = person_id
```

Join Queries

Joining Condition

- For a useful Join query a joining condition is required
  - Defined in where clause as relationships between columns
  - Multiple conditions may be defined if multiple columns shared
  - More than two tables can be joined in a query

Problem: Find people who live in same state as studio

Schema:

Studios(studio_id, studio_state, studio_name, studio_city)

People(person_fname, person_lname, person_id, person_state, person_city)

Query:

```
Select person_fname, person_lname, studio_name
From Movies, People
Where studio_city = person_city
AND studio_state = person_state
```

Join Queries

More than two tables

- Separate condition is required to join each table

Problem: Get title, director, studio, city for all movies in the database

Schema:

Studios(studio_id, studio_state, studio_name, studio_city)

People(person_fname, person_lname, person_id, person_state, person_city)

Movies(movie_title, director_id, studio_id)

Query:

```
Select M.movie_title, M.studio_id, P.person_fname, P.person_lname, S.studio_city
From Movies M, People P, Studio S
Where M.director_id = P.person_id
AND M.studio_id = P.person_id
```

Join Queries
Self Join

- Required to compare values within a single column
  - Need to define aliases for the table names

Problem: Find actors living in the same state

Schema:

People(person_fname, person_lname, person_id, person_state, person_city)

Query:

```
Select p1.person_id, p1.person_fname, p1.person_lname, p1.person_state
From People p1, People p2
Where p1.person_state = p2.person_state
AND p1.person_id != p2.person_id
```

Note: Distinct operator is critical because if there are multiple people from any state each person will appear as many times as there are people from that state

Join Queries
Processing

1. Cartesian product of the two tables involved is taken.
   - Combination of all rows of one table with all rows of the other table
   - 2 tables with 3 and 10 records will have 30 records in the joined table
   - 3 tables with 10, 22, 11 records will have 2420 records in the joined table
2. The where clause is enforced on the resulting table which eliminates all the rows that do not meet the conditions
   - Any sub queries in the where clause are evaluated to allow the results to be used in the where clause.
3. If a group by clause is present the remaining rows of the table are sorted according to the group by columns
4. If aggregate functions are present in the select, they are applied and the working table is replaced by the one with aggregate values
5. Having clause, if present is applied to the groups created using the GROUP clause.
   - Rows that do not conform to the Having clause are discarded.

Join Queries

Union

- Union Joins allow multiple query results to be combined into a single result set

Syntax

```
Select select_list
From table [,table, ….]
[Where condition]
Union [All]
Select select_list
From table [,table, ….]
[Where condition]
```

Example

```
Select person_id, person_city, person_state
 From People
 Union
 Select studio_id, studio_city, studio_state
 From Studios
```

- Notes:
  - The number of columns selected for both the queries should be the same
  - The columns are merged in order in which they are selected
  - The duplicates are eliminated from the combined table
  - More than two tables can be joined together

Join Queries

Union (All & Order By)

- Union query eliminates all duplicates in the resultant table
  - All option is used when we do not want to eliminate the duplicates
- Union and Order By can be used together to order the results of the combined table
  - This clause is not allowed when a single column result is obtained and the all keyword is used since the duplicates are eliminated and there is nothing to order by
- Example

```
Select studio_id, studio_state
From Studios
Union
Select Person_id, person_state
From People
Order By studio_state
```

Join Queries

Intersect

- In the Intersect Query results of two separate queries are concatenated, however, only common elements of the two queries are included in the resultset
- Example

```
Select person_state
From People
Intersect
Select studio_state
From Studios
```

Join Queries

Minus

- Minus Query lists all the records which are present in the first but not in the second.
- **Example**

```
Select person_state
From People
Minus
Select studio_state
From Studios
```

Join Queries
SQL 92 Syntax

- More verbose than pervious versions of SQL
  - Need to define aliases for the table names
- Separates the condition for joining from condition for filtering

Example: Find actors living in the same state

Schema:

People(person_fname, person_lname, person_id, person_state, person_city)
Movies(movie_title, director_id, studio_id)

Query:

```
Select movie_title, person_fname, person_lname
From Movies INNER JOIN People
ON director_id = person_id

Select movie_title, person_fname, person_lname
From Movies INNER JOIN People
ON director_id = person_id
Where studio_id = 1
```

Join Queries

SQL 92 Syntax (Multiple Table Join)

Example: Get title, director, studio, city for all movies in database

Schema:

Studios(studio_id, studio_state, studio_name, studio_city)

People(person_fname, person_lname, person_id, person_state, person_city)

Movies(movie_title, director_id, studio_id)

Query:

```
Select Movies.movie_title, Movies.studio_id, Person.person_fname,
      Person.person_lname, Studio.studio_city
From (People Inner Join
        (Movies Inner Join Studio
         On Studio.studio_id = Movie.studio_id)
         On Movie.director_id = Person.person_id
```

Join Queries

SQL 92 Syntax (Left/Right/Full Join)

Schema:

People(person_fname, person_lname, person_id, person_state, person_city)

Movies(movie_id, movie_title, director_id, studio_id)

Location(movie_id, city, state)

Query:

```
Select movie_title, city, state
From Movies Left Join Locations
        On Movies.movie_id = Locations.movie_id
```

Includes all non matched movie titles

```
Select movie_title, person_fname, person_lname
From Movies Right Join People
        On Movies.director_id = Person.person_id
```

Includes all people not matching to directors

```
Select movie_title, person_fname, person_lname
From Movies Full Join People
        On Movies.director_id = Person.person_id
```

Includes Non matched People and directors

Nested Queries

Definitions

- A nested query is a query inside another query
  - The enclosing query also called outer query
  - Nested query is called inner query
- It usually appears as a condition in where or having clauses.
- There can be multiple levels of nesting
- There are two kinds of nested queries
  - Correlated
  - Non-Correlated

Example:

```
Select movie_title
From movies
Where director_id IN (
                Select person_id
                From People
                Where person_state = 'TX')
```

Nested Queries

Non-Correlated

- Generates data required by outer query before it can be executed
- Inner query does not contain any reference to outer query
- Behaves like a procedure
- The result should not contain any column from the nested query
- Example

Schema: People(person_fname, person_lname, person_id, person_state, person_city)
Movies(movie_id, movie_title, director_id, studio_id)

Query:
```
Select movie_title, studio_id
From Movies
Where director_id IN (Select person_id
                      From People
                      Where person_state = 'TX')
```

Steps:

1. Subquery is executed
2. Subquery results are plugged into the outer query
3. The outer query is processed

Nested Queries

Non-Correlated Query

- A noncorrelated subquery is evaluated before the enclosing query & the data returned by the subquery is used by enclosing query.
  - The nested query is executed first, then the returned values are plugged into the enclosing query, and enclosing query is finally processed.
- A noncorrelated subquery can be spotted since the nested query does not contain references to the enclosing query.
- There are a several ways noncorrelated subqueries can be written, but most common probably uses the IN clause.
- The general syntax of such a query is as follows:

  SELECT *select_list* FROM *table* [, *table*, …] WHERE *column_name* IN (SELECT [DISTINCT] *column* FROM *table* [WHERE *condition*]

Nested Queries

Non-Correlated Query: Example

- Find all the movie titles where the director is from texas.

```
SELECT movie_ title, studio_ id
FROM Movies
WHERE director _id IN (
                        SELECT person _id
                        FROM People
                        WHERE person _state = 'TX' )
```

- In the absence of a nested query two queries would be required as follows

```
SELECT person _id
FROM People
WHERE person _state = 'TX'
SELECT movie _title, studio _id
FROM Movies
WHERE director _id IN (4, 5, 6, 10)
```

Nested Queries

Equivalent Join Query

Example:

People(person_fname, person_lname, person_id, person_state, person_city)
Cast_Movies(cast_member_id, role, movie_id)

Select person_fname, person_lname

From People, Cast_Movies

Where Cast_member_id = person_id

And role = 'Pam Green'

Nested Queries

Correlated

- A correlated sub query requires data returned by the enclosing query before it can be executed.
- Correlated sub query runs once for each row selected by the outer query.
- The sub query is then executed using data received from the enclosing query, and the data returned by the sub query is plugged back into the enclosing query for comparison.
- Sub query contains a reference to a value from the row selected by the outer query.
  - The enclosing and sub query thus behave like a loop

Nested Queries

Correlated: Example

Example:

Schema: People(person_fname, person_lname, person_id, person_state, person_city)

Cast_Movies(cast_member_id, role, movie_id)

Query:

```
Select person_fname, person_lname
    From People p1
    Where 'Pam Green' in ( Select role
                           From Cast_Movies
                           Where p1.person_id = cast_member_id)
```

Steps:

1. Contents of the table row in outer query are read
2. Sub-query is executed using data in the row being processed.
3. Results of the inner query are passed to the where in the outer query
4. The Outer query is Processed
5. Loop continues till all rows are exhausted

Crosstab Queries

Definition

- Crosstab queries analyze one field in a table and view by two or more other fields in a table.
  - i.e. standard aggregate functions, such as sum, count and average can be computed
- Scenarios
  - Crosstab queries can be used to keep track of product sales in certain areas of a country, and you can narrow that search into cities of each of those countries.
  - Outstanding receivables that are 30, 60, or 90 days or more in arrears can be tracked in the same table

Crosstab Queries

Examples

- **Book Database**

```
TRANSFORM COUNT(Title)          ← Value
SELECT Price                    ← Row
FROM Publishers, Books
WHERE Publishers.pubID=Books.PubId
GROUP BY Price                  ← Row
PIVOT PubName;                  ← Column
```

- **Sales Database**

```
Transform Count(*)
Select SalesPersonName
From Orders
Group By SalesPersonName
Pivot CustName
```

- **Student Job Search Database**

```
Transform Count(JobID)
Select ApproxStartSal
From JobOpening
Group By ApproxStartSal
Pivot DegReq
```

Action Queries

Examples

- Queries that change the structure of the database (DDL)
  - Insert Query

```
Insert Into NewBooks
Select ISBN, PubID, Price
From Books
Where  Price > 20
```

  - Delete Query

```
Delete
From Books
Where Price > 20
```

  - Update Query

```
Update Books
Where
    Books.ISBN=NewPrices.ISBN
Set Books.Price = NewPrices.Price
Where books.price != newprices.price
```

  - Append Query

```
Insert Into books
    Select * from newbooks
```

Parameter Queries

Definitions

- A parameter query is a query in which the criteria for selection records are determined when the query is executed rather than when the query is designed.
- When access encounters a variable during execution it attempts to bin the variable to some value. To do this it performs the following.
- First it checks whether the variable is the name of a field or a calculated field in the query.
  1. It attempts to resolve the parameter as a reference to something from the current environment e.g. a value in an open form
  2. If both of the above do not succeed access asks the user for the value using a parameter value dialog box
- By default access expects the value that you put in the box to the literal strings of text and puts double quotes around them.
  - To get around this you need to put square brackets around your parameters.