SQL Fundamentals

SQL Overview

- Data Definition Language (DDL)
- Create/alter/delete tables and their attributes
- Data Manipulation Language (DML)
- Query one or more tables insert/delete/modify tuples in tables
- Triggers and Advanced Constraints
- Actions executed by DBMS on updates and specify complex integrity constraints

Basic Query Structure

```
SELECT [DISTINCT] <column expression list>
FROM <list of tables>
WHERE <predicate>
```

- Specifies columns to be retained in the results
- Specifies cross-product of tables
- Specifies selection conditions on the tables mentioned in the FROM clause
- The DISTINCT keyword ensures the resulting table does not have duplicates (optional)

Projection and Selection

Projection selects specific columns, while selection filters rows based on conditions.

Example Tables:

Movie (name, year, genre)

Name	Year	Genre
Apocalypse Now	1979	War
The God Father	1972	Crime
Planet Earth II	2016	Nature Documentary

ActedIN (actorname, moviename)

Actorname	Moviename
Marlon Brando	Apocalypse Now
Al Pacino	The God Father
Marlon Brando	The God Father

Projection Example:

```
SELECT name, genre
FROM Movies
```

Result:

Name	Genre
Apocalypse Now	War
The God Father	Crime
Planet Earth II	Nature Documentary

Selection Example:

```
SELECT *
FROM Movies
WHERE year > 2000
```

Result:

Name	Year	Genre
Planet Earth II	2016	Nature Documentary

Combined Projection and Selection:

```
SELECT name
FROM Movies
WHERE year > 2000
```

Result:

Name
Planet Earth II

SQL Joins

Types of Joins

- Inner Join
- Self Join
- Outer Join (Left, Right, Full)

Inner Joins

Inner joins combine rows from two or more tables based on a related column.

Example: Return all movie genres that Marlon Brando has acted in

```
SELECT DISTINCT genre
FROM Movie, ActedIN
WHERE Movie.name = ActedIN.moviename AND ActedIN.actorname = 'Marlon Brando'
```

Input Tables:

Name	Year	Genre
Apocalypse Now	1979	War
The God Father	1972	Crime
Planet Earth II	2016	Nature Documentary

Actorname	Moviename
Marlon Brando	Apocalypse Now
Al Pacino	The God Father
Marlon Brando	The God Father

Join Result (intermediate):

Name	Year	Genre	Actorname	Moviename
Apocalypse Now	1979	War	Marlon Brando	Apocalypse Now
The God Father	1972	Crime	Marlon Brando	The God Father

Final Result:

Genre
War
Crime

Self Joins

A self join is a regular join, but the table is joined with itself.

Example: Find name of employees and the name of their managers

```
SELECT e.name, m.name
FROM Employee e, Employee m
WHERE e.managerid = m.eid
```

Input Table:

eid	name	salary	managerid
101	John	50000	103
102	Alice	60000	104
103	Mary	80000	NULL
104	Bob	80000	103

Result:

e.name	m.name
John	Mary
Alice	Bob
Bob	Mary

Outer Joins

Outer joins return all rows from one or both tables, including unmatched rows.

Left Outer Join:

```
SELECT Movie.name, ActedIN.actorname
FROM Movie LEFT OUTER JOIN ActedIN
ON Movie.name = ActedIN.moviename
```

Result:

Movie.name	ActedIN.actorname
Apocalypse Now	Marlon Brando
The God Father	Al Pacino
The God Father	Marlon Brando
Planet Earth II	NULL

Right Outer Join:

```
SELECT Movie.name, ActedIN.actorname
FROM Movie RIGHT OUTER JOIN ActedIN
ON Movie.name = ActedIN.moviename
```

Result:

Movie.name	ActedIN.actorname
Apocalypse Now	Marlon Brando
The God Father	Al Pacino
The God Father	Marlon Brando
NULL	Leonardo DiCaprio

Full Outer Join:

```
SELECT Movie.name, ActedIN.actorname
FROM Movie FULL OUTER JOIN ActedIN
ON Movie.name = ActedIN.moviename
```

Result:

Movie.name	ActedIN.actorname
Apocalypse Now	Marlon Brando
The God Father	Al Pacino
The God Father	Marlon Brando
Planet Earth II	NULL
NULL	Leonardo DiCaprio

SQL Aggregation and Grouping

Aggregate Functions

Five basic aggregate operations in SQL:

- COUNT: counts how many rows are in a particular column
- SUM: adds together all the values in a particular column
- MIN and MAX: return the lowest and highest values in a particular column, respectively
- AVG: calculates the average of a group of selected values

Except COUNT, all aggregations apply to a single attribute.
Examples:

```
SELECT count(*)
FROM Movie

SELECT count(DISTINCT genre)
FROM Movie

SELECT count(genre)
FROM Movie
```

NULL Handling in Aggregates

NULL is ignored in any aggregation (It does not contribute to any aggregate).

Example Table:

Name	Year	Genre	Budget	Revenue	Rate
Pirates of the Caribbean	2007	Action	$300M	$900M	7.1
The Lion King	2019	Animation	$260M	$1.65B	6.5
The Dark Knight	2008	Action	NULL	NULL	9.5
Toy Story 3	NULL	Animation	$300M	$1B	8.3
American Sniper	2013	Action	$59M	$350M	7.3

Examples:

```
SELECT count(*)
FROM Movie
-- Returns 5

SELECT count(year)
FROM Movie
-- Returns 4 (ignores NULL)

SELECT sum(revenue)
FROM Movie
-- Returns sum of non-NULL revenues
```

GROUP BY Operations

GROUP BY groups rows that have the same values into summary rows.

Example: Find the total revenue for all movies produced after 2008 by genre

```
SELECT genre, SUM(revenue) AS TotalRevenue
FROM Movie
WHERE year > 2008
GROUP BY genre
```

Input Table:

Name	Year	Genre	Revenue
The Lion King	2019	Animation	$1.65B
Toy Story 3	2010	Animation	$1B
American Sniper	2013	Action	$350M

Result:

Genre	TotalRevenue
Animation	$2.65B
Action	$350M

Multiple Grouping Attributes:

```
SELECT genre, year, SUM(revenue - budget) AS TotalProfit
FROM Movie
GROUP BY genre, year
```

Important: Everything in SELECT must be either a GROUP BY attribute or an aggregate.

SQL Subqueries and Quantifiers

Subquery Overview

- Subquery: A query that is part of another
- Nested Query: A query that has an embedded subquery
- A subquery can be a nested query itself

A subquery may occur in:

- A SELECT clause
- A FROM clause
- A WHERE clause

Rule of thumb: avoid nested queries when possible (but sometimes it's impossible).

Subqueries in SELECT, FROM, and WHERE

1. Subqueries in SELECT:

```
SELECT a.actorname, (SELECT genre
                     FROM Movie m
                     WHERE m.name = a.moviename) as genre
FROM ActedIn a
```

This is a "correlated subquery" because the inner query references the outer query.

Equivalent unnested query:

```
SELECT a.actorname, genre
FROM ActedIn a, Movie m
WHERE m.name = a.moviename
```

2. Subqueries in FROM:

```
SELECT x.name, rating
FROM (SELECT *
      FROM Movie AS m
      WHERE rating > 8) as x
WHERE x.rating < 9
```

Alternative using WITH:

```
WITH myTable AS (SELECT * FROM Movie AS m WHERE rating > 8)
SELECT x.name, x.rating
FROM myTable as x
WHERE x.rating < 9
```

3. Subqueries in WHERE:
Existential Quantifiers:
Find the name of actors who have acted in some Sci-Fi movie:

```
SELECT DISTINCT a.actorname
FROM ActedIn a
WHERE EXISTS (SELECT m.name
              FROM Movie m
              WHERE m.name = a.moviename AND
                    m.genre = 'Sci-Fi')
```

Using IN:

```
SELECT DISTINCT a.actorname
FROM ActedIn a
WHERE a.moviename IN (SELECT m.name
                      FROM Movie m
                      WHERE m.name = a.moviename AND
                            m.genre = 'Sci-Fi')
```

Universal Quantifiers:
Retrieve all actor names that only acted on action movies:

```
SELECT DISTINCT a.actorname
FROM ActedIn a
WHERE a.actorname NOT IN (SELECT a.actorname
                          FROM Movie m, ActedIn a
                          WHERE m.name = a.moviename AND
                                m.genre != 'Action')
```

Numeric Comparisons:
Retrieve all actor names that acted in at most two action movies:

```
SELECT DISTINCT a.actorname
FROM ActedIn a
WHERE 2 >= (SELECT count(*)
            FROM Movie m
            WHERE m.name = a.moviename AND
                  m.genre = 'Action')
```

Relational Data Model

Data Models Overview

A data model is an abstraction for describing and representing data. The description consists of three parts:

- Structure

- Constraints
- Manipulation

Important Data Models:

- Relational: Data represented as a collection of tables (Most Database Systems)
- Semistructured: Data represented as a tree
- Key-value pairs: Data represented as a dictionary or Hash table (NoSQL database systems)
- Graph
- Array/Matrix (Machine Learning)
- Dataframes

Relational Structure

- Data is a collection of relations
- A relation is a table that consists of a set of tuples or records
- Attribute (Field, Column) is atomic typed data entry

Example Relation:

SID	Name	Surname	Age	GPA
1	Alicia	Shan	20	3.5
2	Andre	Lorde	21	3
3	Yan	Ke	19	4
4	Sudip	Roy	22	4

Relational Schema:

Describes the relation's name, attribute name, and their domain name (meta-data)

Student (sid: string, name: string, surname: string, age: integer, gpa: real)

Key Concepts:

- Tuple (Record, Row): a single entry in the table
- Relational Instance: a set of tuples conforming to the same schema (data)
- Cardinality: the number of tuples in a relation (4 in the example)
- Arity: the number of attributes of a relation (5 in the example)

Integrity Constraints and Keys

Data is only as good as information stored in it. The relational data model allows us to impose various constraints on data.

Integrity Constraints (IC): conditions specified on a database schema that restrict the data that can be stored.

Key Constraint: a statement that a minimal subset of attributes uniquely identify a tuple.

Types of Keys:

- (Candidate) Key: a set of attributes that uniquely identify a tuple
- Super Key: a set of attributes that contain a key
- Primary Key: a database designer identifies one key and designates it as primary key
- Composite Key: a key consisting of multiple attributes

Foreign Key Constraint:
Sometimes data stored in a relation is linked to data stored in another relation. If one of the relations is modified, the other should be checked for consistency.

Example:
Student (sid, name, surname, age, gpa)
Enrolled (cid, sid, grade)

Here, sid in Enrolled is a foreign key referencing the primary key sid in Student.

SQL for Data Definition and Manipulation

Data Definition Language (DDL):

```
CREATE TABLE Students(
  sid CHAR(20),
  name CHAR(30),
  surname CHAR(20),
  age INTEGER
)

ALTER TABLE Student
ADD Email varchar(255)

DROP TABLE Student
```

Data Manipulation Language (DML):

```
INSERT INTO Students (sid, name, surname, age)
VALUES ('1', 'Alicia', 'Shan', 20)
```

```
UPDATE Students
SET gpa = gpa + 0.5
WHERE name = 'Alicia'

DELETE FROM Students
WHERE name = 'Ziaho'
```

First Normal Form (1NF):
All relations must be flat: we say that the relation is in first normal form.

Example of 1NF:
Instead of storing courses as a nested structure:

SID	Name	Surname	Age	GPA
1	Alicia	Shan	20	3.5
2	Andre	Lorde	21	3

CID	SID	Grade
dsc100	1	97
dsc80	1	90
dsc100	2	91

DataFrame Data Model

Origins and Characteristics

- 1992: Emerged in S programming language at Bell Labs
- 2000: Inherited by R programming language
- 2009: Brought to Python by Pandas

DataFrames support relational operators (e.g., filter, join), linear algebra (e.g., transpose), and spreadsheet-like (e.g., pivot) operators.

Example DataFrame:

Name	FName	City	Age	Salary
Smith	John	3	35	$280
Doe	Jane	1	28	$325
Brown	Scott	3	41	$265
Howard	Shemp	4	48	$359
Taylor	Tom	2	22	$250

Comparison with Relational Model

In Comparison to Relational Tables:

- Lazily-induced schema
- Rows are named and ordered
- Heterogeneous data types

In Comparison to Matrices:

- Rows and columns are labeled
- Columns and rows equivalent

SQL Core Concepts

Summary

SQL (Structured Query Language) is the standard language for interacting with relational databases. It supports:

- Data Definition Language (DDL): Creating, altering, and deleting tables and attributes
- Data Manipulation Language (DML): Inserting, updating, deleting, and querying data
- Querying: Using SELECT statements with projection, selection, joins, grouping, and aggregation
- Constraints: Enforcing data integrity via keys and foreign keys

Core Concept Examples

Basic Aggregation and GROUP BY:

```
SELECT genre, SUM(revenue) AS TotalRevenue
FROM Movie
WHERE year > 2008
GROUP BY genre
```

HAVING Clause (Derived Example):

```
SELECT genre, COUNT(*) AS MovieCount
FROM Movie
GROUP BY genre
HAVING COUNT(*) > 2
```

This query returns genres with more than two movies.

Multiple Aggregates with HAVING (Derived Example):

```
SELECT genre, AVG(rating) AS AvgRating, SUM(revenue) AS TotalRevenue
FROM Movie
GROUP BY genre
HAVING AVG(rating) > 8.0
```

Returns genres where the average rating is above 8.0.

Key Points

- SQL queries can project, filter, join, group, and aggregate data
- GROUP BY is used to aggregate data by one or more columns
- HAVING filters groups after aggregation
- Aggregates ignore NULL values
- All columns in SELECT must be either grouped or aggregated

DataFrame Core Concepts

Summary

DataFrames are a tabular data structure supporting relational, linear algebra, and spreadsheet-like operations. They are widely used in data science and analytics, with origins in S, R, and Python's pandas library.

Core Concept Examples

Basic DataFrame Operations (Derived Example):

```
# Filtering rows where Age > 30
df_filtered = df[df['Age'] > 30]

# Grouping and aggregating
df_grouped = df.groupby('City')['Salary'].mean()
```

Advanced DataFrame Operations (Derived Example):

```
# Pivot table: Average Salary by City and Age Group
df['AgeGroup'] = pd.cut(df['Age'], bins=[20, 30, 40, 50],
                        labels=['20-30', '31-40', '41-50'])
pivot = df.pivot_table(values='Salary', index='City',
                       columns='AgeGroup', aggfunc='mean')

# Merging DataFrames (similar to SQL JOIN)
merged = pd.merge(df1, df2, left_on='SID', right_on='SID', how='inner')
```

Key Points

- DataFrames support relational (filter, join), linear algebra (transpose), and spreadsheet-like (pivot) operations
- Schema is often inferred from data (lazily-induced)
- Rows and columns are labeled and can be heterogeneous

- Advanced operations include pivot tables, merges (joins), and groupby-aggregate patterns