SQL

- 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 SQL Query

```
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 resulting table should not have duplicates (it's optional)

Projection in SQL

Movie (name, year, genre) ActedIN (actorname, moviename)

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

Return movies names and their genres

"Name"	"Year"	"Genre"
"Apocalypse Now"	"1989"	"War"
"The God Father"	"1972"	"Crime"
"Planet Earth II"	"2016"	"Nature Documentary"

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

Selection in SQL

Movie (name, year, genre) ActedIN (actorname, moviename)

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

"Name"	"Year"	"Genre"
"Apocalypse Now"	"1989"	"War"
"The God Father"	"1972"	"Crime"
"Planet Earth II"	"2016"	"Nature" , "Documentary"

Return movies produced after 2000

"Name"	"Year"	"Genre"
"Planet Earth II"	"2016"	"Nature" , "Documentary"

Selection and Projection in SQL

Movie (name, year, genre)

"Name"	"Year"	"Genre"
"Apocalypse Now"	"1989"	"War"
"The God Father"	"1972"	"Crime"
"Planet Earth II"	"2016"	"Nature Documentary"

ActedIN (actorname, moviename)

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

Return movie names produced after 2000

Name

Planet Earth II

What does this query return? Joins in SQL

```
SELECT DISTINCT genre
FROM Movie, ActedIN
WHERE movie.name=ActedIN.moviename
```

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

Joins in SQL

- Inner Join
- Self Join
- Outer Join

Joins

Return all movie genres that Marlon Brando has acted in

Movie (name, year, genre) ActedIN (actorname, moviename) Moviename = foreign key to Movie.name

(Inner) Joins

Join Condition

Movie (name, year, genre) ActedIN (actorname, moviename)

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

(Inner) Joins

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

"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"

(Inner) Joins

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

"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"

(Inner) Joins

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

"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"

(Inner) Joins

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

"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"

"Name"	"Year"	"Genre"	"Actorname"	"Moviename"
"Apocalypse Now"	"1989"	"War"	"Marlon Brando"	"Apocalypse Now"

(Inner) Joins

```
SELECT DISTINCT genre
FROM   Movie, ActedIN
```

```
WHERE  Movie.name=ActedIN.moviename AND ActedIN.actorname='Marlon Brando'
```

"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"

"Name"	"Year"	"Genre"	"Actorname"	"Moviename"
"Apocalypse Now"	"1989"	"War"	"Marlon Brando"	"Apocalypse Now"

(Inner) Joins

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

"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"

"Name"	"Year"	"Genre"	"Actorname"	"Moviename"
"Apocalypse Now"	"1989"	"War"	"Marlon Brando"	"Apocalypse Now"

(Inner) Joins


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


Movie (name, year, genre) ActedIN (actorname, moviename)

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

Name	Year	Genre	Actorname	Moviename
Apocalypse Now	1989	War	Marlon Brando	Apocalypse Now

(Inner) Joins

Movie (name, year, genre) ActedIN (actorname, moviename)

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

Name	Year	Genre	Actorname	Moviename
Apocalypse Now	1989	War	Marlon Brando	Apocalypse Now
The God Father	1972	Crime	Marlon Brando	The God Father

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

(Inner) Joins

Movie (name, year, genre) ActedIN (actorname, moviename)

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

Name	Year	Genre	Actorname	Moviename
Apocalypse Now	1989	War	Marlon Brando	Apocalypse Now
The God Father	1972	Crime	Marlon Brando	The God Father

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

(Inner) Joins

Movie (name, year, genre) ActedIN (actorname, moviename)

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

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

(Inner) Joins

Movie (name, year, genre) ActedIN (actorname, moviename)

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

Self Join

Employee (eid, name, salary, managerid)

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
```

Self Join

Employee (eid, name, salary, managerid)

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

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

Outer Joins

- Left Outer Join
- Right Outer Join
- Full Outer Join

Outer Joins

Left Outer Join

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

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
```

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
```

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

Joins on More Than Two Tables

```
SELECT DISTINCT name, genre, actorname
FROM   Movie, ActedIN
WHERE  Movie.name= ActedIN.moviename AND Movie.year > 1975
```

Name	Genre	Actorname
Apocalypse Now	War	Marlon Brando

```
SELECT DISTINCT name, genre, actorname
FROM   Movie, ActedIN
WHERE  Movie.name= ActedIN.moviename AND Movie.year > 1975
```

"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"	"$185M"	"$1B"	"9.5"
"Toy Story 3"	"2010"	"Animation"	"$300M"	"$1B"	"8.3"
"American Sniper"	"2013"	"Action"	"$59M"	"$350M"	"7.3"

What type of summary statistics could be of interest?

Aggregate Functions

Five basic aggregate operations in SQL

COUNT counts how many rows are in a particular column. [cite: 61] SUM adds together all the values in a particular column. [cite: 62, 63, 64] MIN and MAX return the lowest and highest values in a particular column, respectively. [cite: 62, 63, 64] AVG calculates the average of a group of selected values. [cite: 62, 63, 64]

Except count, all aggregations apply to a single attribute

EXAMPLE

```
SELECT count (*)
FROM Movie
```

```
SELECT count (DISTINCT genre)
FROM   Movie

SELECT count (genre)
FROM   Movie
```

We probably want this

Aggregates and NULL Values

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

"Name"	"Year"	"Genre"	"Budget"	"Revenue"
"Pirates of the Caribbean"	"2007"	"Action"	"$300M"	"$900M"
"The Lion King"	"2019"	"Animation"	"$260M"	"$1.65B"
"The Dark Knight Toy Story 3"	"2008 2010"	"Action Animation"	"$185M $300M"	"$1B $1B"
"American Sniper"	"2013"	"Action"	"$59M"	"$350M"

Aggregates and NULL Values

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

"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"

```
select count (*)
from   Movie

select count (*)
from   Movie

select count (year)
from   Movie

select sum (revenue)
from   Movie
where  revenue is not null
```

Aggregates and NULL Values

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

"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"

```
select count (*)
from   Movie

select count (*)
from   Movie

select count (year)
from   Movie

select sum (revenue)
from   Movie
where  revenue is null
```

Grouping and Aggregation

Movie (name, year, genre, budget, rate, revenue)

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

"Name"	"Year"	"Genre"	"Revenue"
"Pirates of the Caribbean"	"2007"	"Action"	"$900M"
"The Lion King"	"2019"	"Animation"	"$1.65B"
"The Dark Knight"	"2008"	"Action"	"$1B"
"Toy Story 3"	"2010"	"Animation"	"$1B"
"American Sniper"	"2013"	"Action"	"$350M"

```
SELECT genre, Sum(revenue) AS Total Revenue
FROM   Movie
WHERE  year > 2008
GROUP BY genre
```

"Name"	"Year"	"Genre"	"Revenue"
"Toy Story 3"	"2010"	"Animation"	"$900M"
"The Lion King"	"2019"	"Animation"	"$1.65B"
"Pirates of the Caribbean"	"2007"	"Action"	"$1B"
"The Dark Knight"	"2008"	"Action"	"$1B"
"American Sniper"	"2013"	"Action"	"$350M"

```
SELECT genre, Sum(revenue) AS Total Revenue
FROM   Movie
WHERE  year > 2008
GROUP BY genre
```

Grouping and Aggregation Name Year Genre Revenue Toy Story 3 2010 Animation $900M The Lion King 2019 Animation $1.65B Pirates of the Caribbean 2007 Action $1B The Dark Knight 2008 Action $1B American Sniper 2013 Action $350M SELECT genre, Sum(revenue) AS TotalRevenue FROM Movie WHERE year ¿ 2008 GROUP BY genre

Genre	Total Revenue
Animation	$2.65B
Action	$1.35B

Other Examples

Compare these two queries: SELECT genre, Sum(revenue) AS TotalRevenue FROM Movie GROUP BY genre SELECT genre, Sum(revenue) AS TotalRevenue FROM Movie GROUP BY year One answer for each year One answer for each genre Other Examples SELECT year sum(budget) AS SumBudget, max(revenue) AS MaxRevenue FROM movie GROUP BY year Mix and match aggregates Multiple Aggregates SELECT genre, Sum(revenue - budget) AS TotalProfit FROM Movie GROUP BY genre, year Multiple grouping attribute Other Examples Name Year Genre Revenue Toy Story 3 2010 Animation $900M The Lion King 2019 Animation $1.65B Pirates of the Caribbean 2007 Action $1B The Dark Knight 2008 Action $1B American Sniper 2013 Action $350M Genre Revenue Animation $2.55B Action $2.35B SELECT genre, Sum(revenue) FROM Movie WHERE year ¿ 2008 GROUP BY genre Other Examples Everything in SELECT must be either a GROUP BY attribute, or an aggregate Name

What We Have Learned So Far

- Data models
- Relational data model
- Structure
- Complaints
- Manipulation: SQL

What We Have Learned So Far

SQL Features

- Projections
- Selections
- Joins (inner and outer)
- Group by
- Having
- Inserts, updates,
- Aggregates
- and deletes

Subqueries

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

Why? Sometimes we need to express a condition that refers to a table that must itself be computed

A subquery may occur in:

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

Often appear here
Rule of thumb: avoid nested queries when possible
(But sometimes it's impossible, as we will see)

Subqueries

- Can return a single value to be included in a SELECT clause
- Can return a relation to be included in the FROM clause
- Can return a single value to be compared with another value in a WHERE clause
- Can return a relation to be used in the WHERE or HAVING

1. Subqueries in SELECT

Movie(name, year, genre, budget, revenue, rating)
ActedIN (actorname, moviename, salary)

1.Subqueries in SELECT

Movie (name, year, genre, budget, revenue, rating) ActedIN (actorname, moviename, salary)

For each actor return the genre of movie they acted in

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

"Correlated subquery"

What happens if the subquery returns more than one genre?

1.Subqueries in SELECT

Movie (name, year, genre, budget, revenue, rating) ActedIN (actorname, moviename, salary)

Whenever possible, don't use a nested queries:

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

```
"SELECT
","a.actorname, genre
"
"FROM
","ActedIn a, Movie m
```

```
"
"WHERE
","m.name = a.moviename
"
```

Subquery unnesting

1.Subqueries in SELECT

Movie (name, year, genre, budget, revenue, rating) ActedIN (actorname, movienname, salary)

Compute average salary of actors for all movies with rating ¿9

```
SELECT DISTINCT m.name, (SELECT AVG (salary)
FROM   ActedIn a
WHERE  m.name = a.moviename) as salary
FROM   Movie.m
WHERE  m.rating >9
```

```
"SELECT
","m.name, AVG(salary)
"
"FROM
","Movie m, ActedIn a
"
"WHERE
","m.name=a.moviename
"
"AND
","m.rating> 9
"
"GROUP BY
","m.name
"
```

Subquery unnesting

1.Subqueries in SELECT

Movie (name, year, genre, budget, revenue, rating) ActedIN (actorname, movienname, salary)

Compute the number of actors in each movie

```
SELECT DISTINCT m.name, (SELECT count (*)
FROM   ActedIn a
WHERE  m.name = a.moviename) as anum
FROM   Movie.m
```

?

```
SELECT
FROM   m.name, count(*)
       ActedIn a, Movie m
WHERE  m.name=a.moviename
GROUP BY m.name
```

Subquery unnesting

1.Subqueries in SELECT

But are these equivalent?

```
SELECT DISTINCT m.name, (SELECT AVG (salary)
FROM   ActedIn a
WHERE  m.name = a.moviename ) as salary
FROM   Movie.m
WHERE  m.rating >9
```

Movie (name, year, genre, budget, revenue , rating) ActedIN (actorname, moviename, salary)

```
SELECT m.name, AVG(salary)
FROM   Movie m, ActedIn a
WHERE  m.name=a.moviename AND m.rating> 9
GROUP BY m.name
=
SELECT m.name, AVG(salary)
FROM   Movie m, LEFT OUTER JOIN ActedIn a
ON     m.name=a.moviename
WHERE  rating> 9
GROUP BY m.name
=
```

2. Subqueries in FROM

Movie(name, year, genre, budget, revenue , rating) ActedIN (actorname, movienname, salary)

2.Subqueries in FROM

Find all Movie with rating ¿ 8 and ¡ 9 "Not a correlated subquery"

Movie (name, year, genre, budget, revenue , rating) ActedIN (actorname, moviename, salary)

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

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

A subquery whose result we called myTable Sub-query refactoring

2.Subqueries in FROM

Find all Movie with rating ¿ 8 and ¡ 9 Movie (name, year, genre, budget, revenue , rating) ActedIN (actorname, moviename, salary)

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

```
SELECT m.name, rating
FROM   myTable as X
WHERE  m.rating < 9 AND m.rating > 8
=
```

Subquery unnesting

3. Subqueries in WHERE

Movie(name, year, genre, budget, revenue , rating) ActedIN (actorname, movienname, salary)

3.Subqueries in WHERE

Find the name of actors who have acted in some Sci-Fi movie

- Existential Quantifiers
- Quantifier is a logical operator that specifies how many elements in the domain of discourse satisfy a property
- "There exists," "there is at least one," or "for some"

Movie (name, year, genre, budget, revenue , rating) ActedIN (actorname, moviename, salary)

```
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')
```

TRUE if the subquery Using EXISTS returns one or more records

3.Subqueries in WHERE

Find the name of actors who have acted in some Sci-Fi movie Movie (name, year, genre, budget, revenue , rating) ActedIN (actorname, moviename, salary)

```
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')
```

Allow us to test set Using IN membership Existential Quantifiers

3.Subqueries in WHERE

Find the name of actors who have acted in some Sci-Fi movie Movie (name, year, genre, budget, revenue , rating) ActedIN (actorname, moviename, salary)

```
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')
```

Existential Quantifiers

```
SELECT DISTINCT a.actorname Subquery unnesting
FROM   Movie m, ActedIN a
WHERE  m.name=a.moviename AND m.genre='Sci-Fi'
```

3.Subqueries in WHERE

Find the name of actors who have acted in some non-Sci-Fi movie Movie (name, year, genre, budget, revenue , rating) ActedIN (actorname, moviename, salary)

```
SELECT DISTINCT a.actorname
FROM   ActedIn a
WHERE  a.moviename NOT IN (SELECT m.name
                           FROM   Movie m
                           WHERE  m.name=a.moviename AND
                                  m.genre='Sci-Fi')
```

Existential Quantifiers

```
SELECT DISTINCT a.actorname
FROM   movie m, ActedIN a
WHERE  m.name=a.moviename AND m.genre $\neq$'Sci-Fi'
```

Subquery unnesting

Existential Quantifiers

are easy J

Join queries essentially check for existential quantifiers

Universal Quantifiers are hard L

The SQL constructs we have discussed so far do not capture universal quantifiers

GOOD NEWS BAD NEWS

3.Subqueries in WHERE

Retrieve all actor names that only acted on action movies

- Universal Quantifiers
- "Given any," "for all," or "for every"
- Same as every movies they acted on were action

Movie (name, year, genre, budget, revenue , rating) ActedIN (actorname, moviename, salary)

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

- Step 1: Find all actor names that acted on some non-action movie
- Step 2: Retrieve all the others (i.e., those do not satisfy the result of Step 1.)

3.Subqueries in WHERE

Retrieve all actor names that only acted on action movies Movie (name, year, genre, budget, revenue , rating) ActedIN (actorname, moviename, salary)

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

- Step 1: Find all actor names that acted on some non-action movie
- Step 2: Retrieve all the others i.e., those do not satisfy the result of Step 1.
- Universal Quantifiers
- "Given any," "for all," or "for every"
- Same as every movies they acted on were action

3.Subqueries in WHERE

Retrieve all actor names that acted on at most two action movies Movie (name, year, genre, budget, revenue , rating) ActedIN (actorname, moviename, salary)

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

What does this query do?

3.Subqueries in WHERE

```
SELECT DISTINCT a.actorname
FROM   ActedIn a
WHERE  0 < (SELECT count (*)
            FROM   Movie m
            WHERE  m.name=a.moviename AND
                   m.genre = 'Action')
```

Find all movie s.t.

3.Subqueries in WHERE

all their actors' salaries ¿ $100K Movie (name, year, genre, budget, revenue , rating) ActedIN (actorname, moviename, salary)

```
SELECT m.name
FROM   movie m
WHERE  $100K < ALL (SELECT a.salary
                    FROM   ActedIn a
                    WHERE  m.name=a.moviename)
```

Not supported in SQLite Universal Quantifiers Is it possible to unnest the universal quantifier query?

Unnesting Universal

Is it possible to unnest the universal quantifier query?

Unnesting Universal

- A query Q is monotone if:
- Whenever we add tuples to one or more input tables, the answer to the query will not lose any output tuple.

Unnesting Universal Quantifiers

```
SELECT a.actorname
FROM   Movie m, ActedIn a
WHERE  m.name=a.moviename AND m.gendre='Crime'
```

Is this monotone?

Monotone Queries

Name	Year	Genre
Apocalypse Now	1989	War
The God Father	1972	Crime
Planet Earth II	2016	Nature Documentary
Jack and Jill	2011	Comedy

Actorname
Marlon Brando
Al Pacino

```
SELECT a.actorname
FROM   Movie m, ActedIn a
WHERE  m.name=a.moviename AND m.gendre='Crime'
```

Data Models

A data model is an abstraction for describing and representing data[cite: 2].
The description consists of three parts:

- Structure
- Constraints
- Manipulation

Important Data Models

- Relational: Data represented as a collection of tables
- Semistructured: Data represented as a tree
- Key-value pairs: Data represented as a dictionary or Hash table
- Graph
- Array/Matrix
- Dataframes

Most Database Systems (Our focus) [cite: 3, 4]
NoSQL database systems [cite: 5]
Machine Learning [cite: 6]

The Relational Data Model

- Structure
- Constraints
- Manipulation

The Relational Data Model

Data is a collection of relations[cite: 9].

A relation is a table that consists of a set of tuples or records[cite: 9].

The Relational Data Model

Attribute (Field, Column) is atomic typed data entry[cite: 13, 14, 15].

Attribute domain

Attribute name

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

Atomic Types

Characters: CHAR(20), VARCHAR(50)

Numbers: INT, BIGINT, SMALLINT, FLOAT

Others: MONEY, DATETIME

Integer

Relational Schema

Describes the relation's name, attribute name, and their domain name (metadata)

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

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

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

Tuple (Record, Column) is a single entry in the table[cite: 16].

Relational Instance

Is a set of tuples conforming to the same schema (data) [cite: 16]

Cardinality is the number of tuples in a relation[cite: 17].
Arity is the number of attributes of a relation[cite: 17].
Arity = 5
Cardinality = 4

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

The Relational Data Model

- Structure
- Constraints
- Manipulation

Integrity Constraints

Data is only as good as information stored in it[cite: 19].

The relational data model allows us to impose various constraints on data[cite: 19].

Integrity Constraints (IC): is a condition specified on a database schema and restrict the data that can be stored in an instance[cite: 19].

We've already discussed one IC: Domain Constraints! [cite: 19]

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

Key Constraint: a statement that a minimal subset of attributes uniquely identify a tuple[cite: 20, 21, 22, 23].

(Candidate) Key: a set of attributes that uniquely identify a tuple[cite: 20, 21, 22, 23].

Not a key

Is this a key?

Composite Key What does it mean?

Key

No two students have the same ID

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

Super Key: a set of attributes that contain a key[cite: 24, 25].

A relation may have several candidate keys[cite: 24, 25].

Primary Key: a database designer identify one key and designate it as primary key[cite: 24, 25].

Key

Another Key

Student (sid, name, surname, age, gpa)

Primary key = sid

Student (sid, name, surname, age, gpa)

Primary key = name, username

Student (sid, name, surname, age, gpa)

Enrolled (cid, sid, grade)

Sometimes data stored in a relation is linked to data stored in another relation[cite: 26].

If one of the relations are modified the other should be checked for consistency[cite: 26].

Foreign Key Constraint

Foreign Key Constraint

Student (sid, name, username, age, gpa)

Enrolled (cid, sid, grade)

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

CID	SID	Grade
dsc100	1	92
dsc80	2	90

Student

Enrolled

It can have a different name

The Relational Data Model

- Structure
- Constraints
- Manipulation

Query Language

Specialized languages for asking questions, or queries from relational data[cite: 29].

- Commercial: SQL (Structured Query Language) [cite: 29]
- Formal: Relational algebra, Relational calculus [cite: 29]

SQL
Data Definition Language (DDL)
Create / alter / delete tables and their attributes - discussed next! [cite: 30]
Manipulating Schema
Manipulating Data Data Manipulation Language (DML) [cite: 31]

SQL: Quick Overview

The CREATE TABLE statement is used to create a new table in a database[cite: 32].

```
CREATE TABLE table_name (
attribute1 type,
attribute2 type,
attribute3 type,
....
)
```

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

SQL: Quick Overview

The INSERT INTO statement is used to insert new records in a table[cite: 33].

```
INSERT INTO table_name (column1, column2, ...)
VALUES (value1, value2, ...)
```

```
INSERT INTO table_name
VALUES (value1, value2, ...)
```

Drop column names if you add values to all columns (be careful with the order)

SQL: Quick Overview

The DELETE statement is used to delete existing records in a table[cite: 34].

```
DELETE FROM table_name WHERE condition
```

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

SQL: Quick Overview

The UPDATE statement is used to modify the existing records in a table[cite: 35, 36].

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

```
UPDATE Students
SET gpa=gpa+3
WHERE name='Ziaho'
```

SQL: Quick Overview

The DROP TABLE statement is used to drop an existing table in a database[cite: 37, 38].

```
DROP TABLE table_name;
```

```
DROP TABLE Student;
```

SQL: Quick Overview

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table[cite: 39, 40].

```
ALTER TABLE table_name;
```

```
ADD column1 type, column2 type,...
ALTER TABLE Student
ADD Email varchar(255)
```

SQL: Quick Overview

The SELECT statement is used to select data from a database[cite: 41, 42].

```
SELECT column1, column2, ...
FROM table_name;
```

```
SELECT name, gpa
FROM student
```

SQL: Quick Overview

The WHERE clause is used to filter records[cite: 43, 44].

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

```
SELECT *
FROM student
WHERE age < 22;
```

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

How would you implement this? [cite: 45]

The logical definition of the data remains unchanged, even when we make changes to the actual implementation[cite: 46].

Physical Data Independence

All relations must be flat: we say that the relation is in first normal form[cite: 47].

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

First Normal Form

How can we store nested information? e.g., suppose we want to add courses enrolled by each student [cite: 48]

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

All relations must be flat: we say that the relation is in first normal form[cite: 48].

E.g., we want to add courses enrolled by each student [cite: 49, 50]

SID	Name	Surname	Age	Enrolled
1	Alicia	Shan	20	
2	Andre	Lorde	21	

CID	Grade
dsc100	97
dsc80	90

CID	Grade
dsc100	91

Non-1NF!

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

Student

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

Enrolled

Now it's in 1NF [cite: 51]

Data Models Summary

Structure + Constraints + Manipulation [cite: 52]

Relational Model:

- Database = collection of tables [cite: 52]
- Each table is flat: "first normal form" [cite: 52]
- Key: may consist of multiple attributes [cite: 52]
- Foreign key: "Semantic pointer" [cite: 52]
- Physical data independence [cite: 52]

The DataFrame Data Model

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

The DataFrame Data Model

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

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

The DataFrame Model

In Comparison to Relational Tables:

- Lazily-induced schema
- Rows are named and ordered
- Heterogenous

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 (Lines: 9-11, 32-40)
- Data Manipulation Language (DML): Inserting, updating, deleting, and querying data (Lines: 12-13, 33-36, 41-44)
- Querying: Using SELECT statements with projection, selection, joins, grouping, and aggregation (Lines: 20-26, 248-266, 296-340)
- Constraints: Enforcing data integrity via keys and foreign keys (Lines: 671-700)

Core Concept Example

Basic Aggregation and GROUP BY

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

Finds the total revenue for all movies produced after 2008, grouped by genre. (Lines: 296-340)

Derived Example: HAVING Clause

The HAVING clause is used to filter groups after aggregation.

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

This query returns genres with more than two movies. **Note: This is a derived example based on the GROUP BY and aggregation principles in Lines: 296-340. The HAVING clause is not explicitly shown in the source.**

Derived Example: Multiple Aggregates with HAVING

```
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. **Note: Derived from aggregation and grouping concepts in Lines: 248-340.**

Key Points

- SQL queries can project, filter, join, group, and aggregate data (Lines: 20-26, 248-340)
- GROUP BY is used to aggregate data by one or more columns (Lines: 296-340)
- HAVING filters groups after aggregation (Derived from SQL standards; not explicitly in the document)
- Aggregates ignore NULL values (Lines: 267-295)
- All columns in SELECT must be either grouped or aggregated (Lines: 296-340)

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. (Lines: 702-720)

Core Concept Example

Basic DataFrame Operations

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

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

These operations correspond to SQL selection and aggregation. (Derived from DataFrame features in Lines: 702-730)

Derived Example: Advanced DataFrame Operations

```
# 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')
```

These advanced operations illustrate DataFrame capabilities beyond basic SQL.
Note: These are derived examples based on DataFrame principles in Lines: 702-730.

Key Points

- DataFrames support relational (filter, join), linear algebra (transpose), and spreadsheet-like (pivot) operations (Lines: 702-720)
- Schema is often inferred from data (lazily-induced) (Lines: 721-730)
- Rows and columns are labeled and can be heterogeneous (Lines: 721-730)
- Advanced operations include pivot tables, merges (joins), and groupby-aggregate patterns (Derived from DataFrame model description)