The Relational Model and SQL

Ramakrishnan & Gehrke Ch 3, 5

(slides adapted from content by J.Gehrke, J.Shanmugasundaram, and/or C.Koch)

Why the Relational Model?

- Extremely common model in DBMSes
  - IBM (DB2), Microsoft (SQLServer), Oracle (Oracle), Sybase, etc...
- Not the only model out there
  - XML or JSON (Hierarchical Data)
    - DB2, Oracle, SQLServer, JAQL, MongoDB
  - RDF (Graph Data)
    - SPARQL Databases

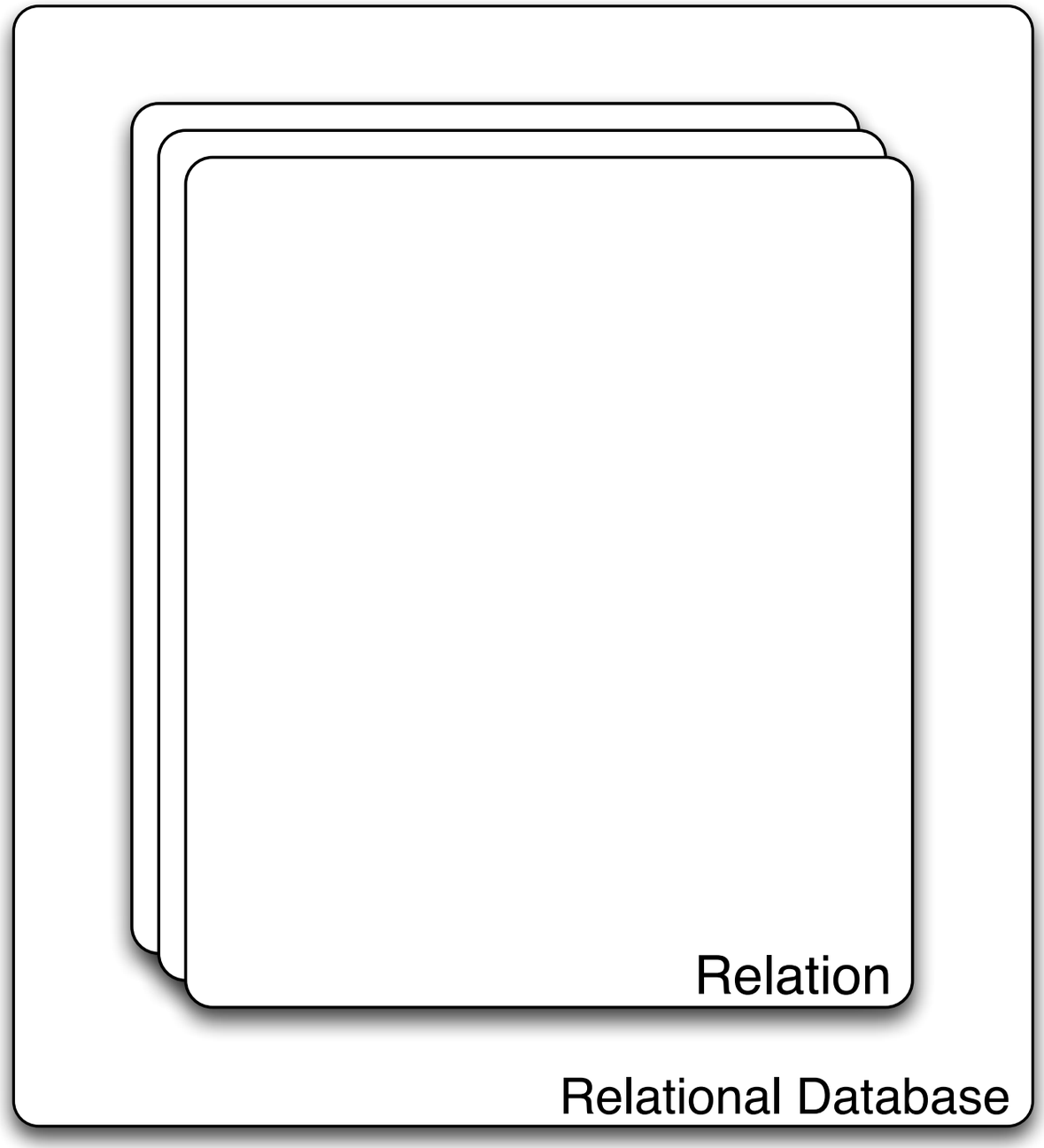
Why the Relational Model?

- Extremely common model in DBMSes
  - IBM (DB2), Microsoft (SQLServer), Oracle (Oracle), Sybase, etc...
- Not the only model out there
  - XML or JSON (Hierarchical Data)
    - DB2, Oracle, SQLServer, JAQL, MongoDB
  - RDF (Graph Data)
    - SPARQL Databases

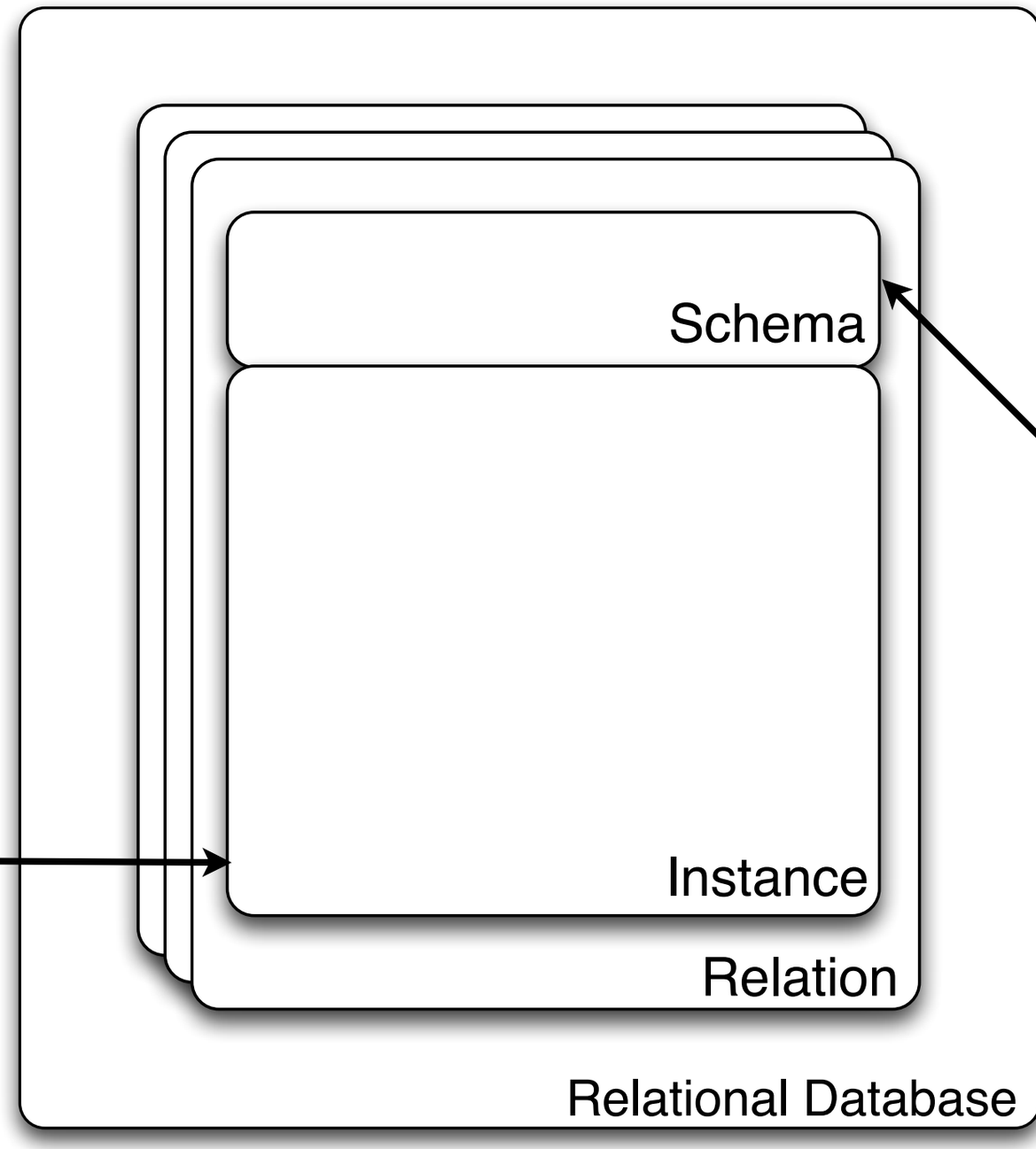
The Relational Model is **Simple**

What is a Relational Database?

What is a Relational Database?



What is a Relational Database?



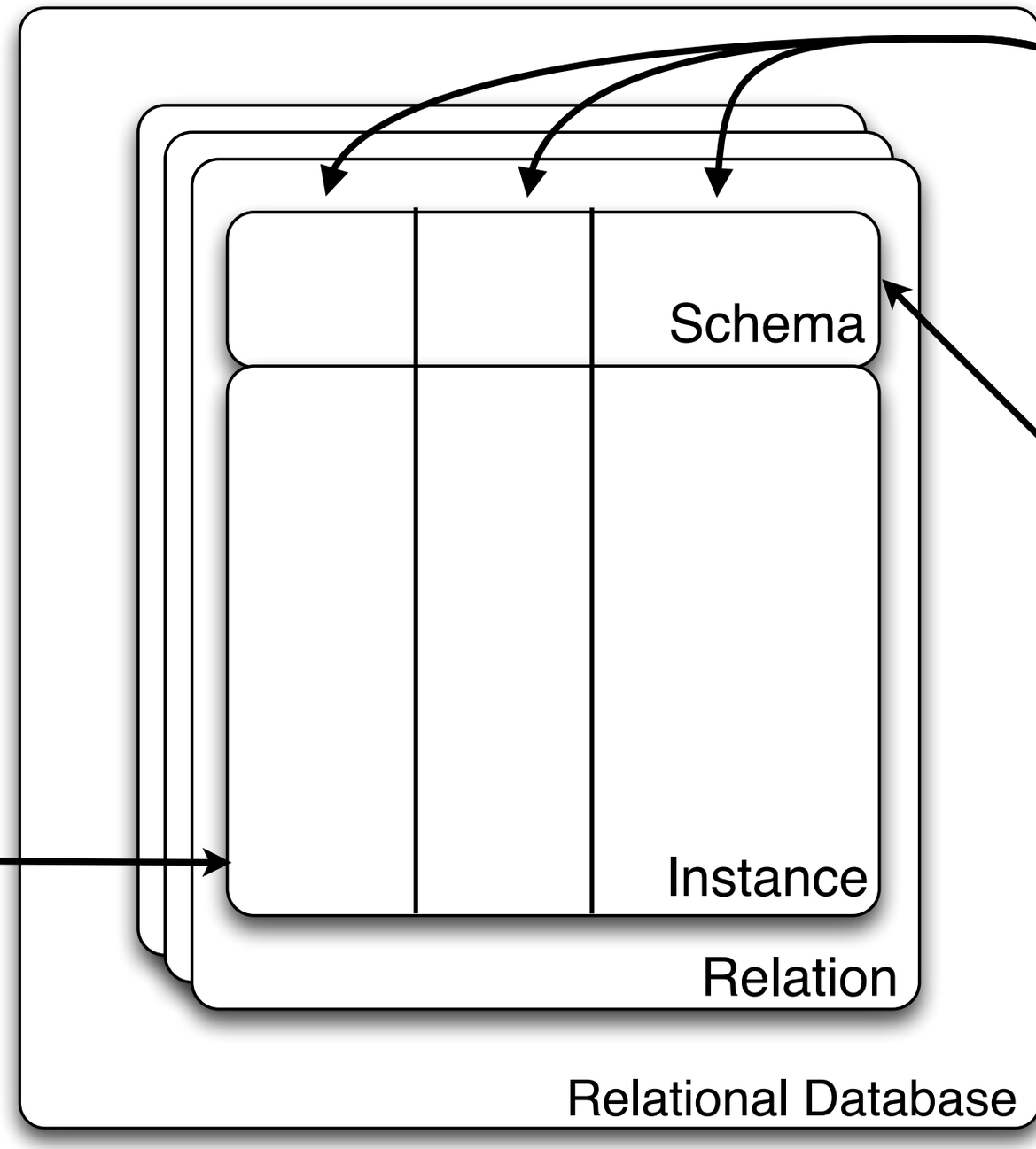
Specifies the name of the relation, plus the name and type of each column

```
Officers(
  firstname: string,
  lastname: string,
  id: int
)
```

The Data

```
[James,    Kirk,   2260]
[Jean Luc, Picard, 2360]
[Benjamin, Sisko,  2365]
```

What is a Relational Database?



Columns
(# = degree/arity)

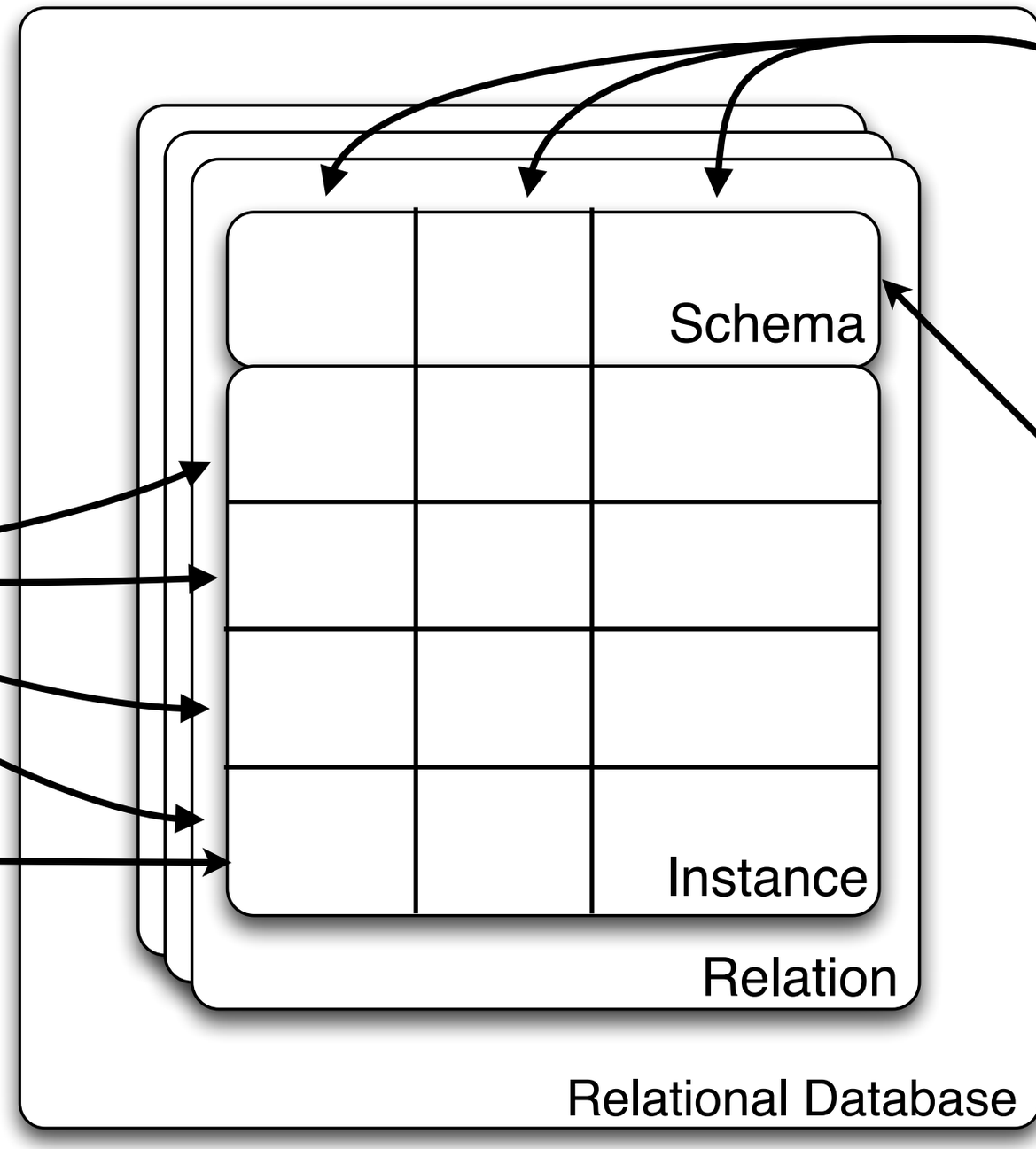
Specifies the name of the relation, plus the name and type of each column

The Data

```
[James,    Kirk,   2260]
[Jean Luc, Picard, 2360]
[Benjamin, Sisko,  2365]
```

```
Officers(
  firstname: string,
  lastname: string,
  id: int
)
```

What is a Relational Database?



Rows
(# = cardinality)

Columns
(# = degree/arity)

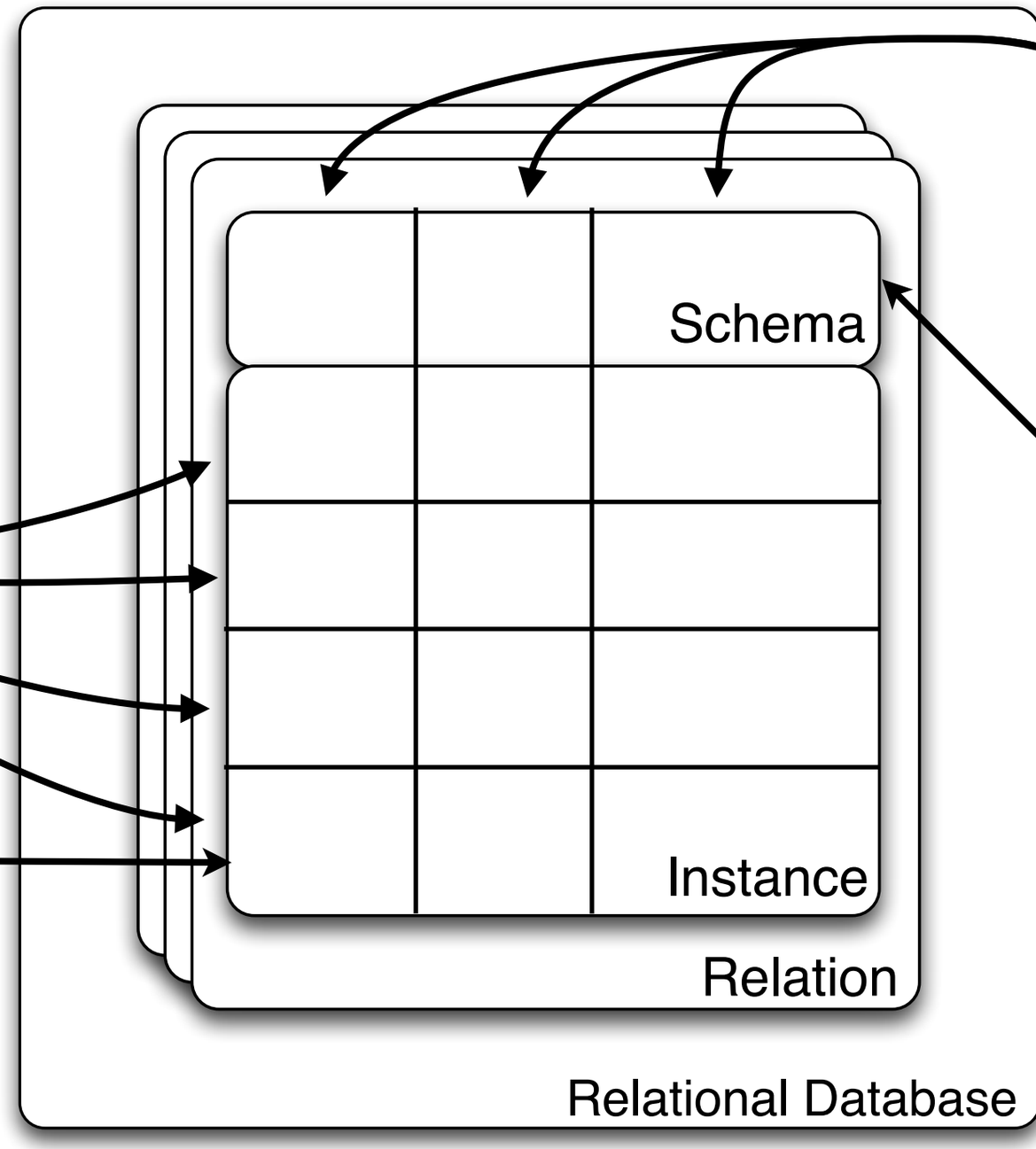
Specifies the name of the relation, plus the name and type of each column

The Data

```
[James,    Kirk,   2260]
[Jean Luc, Picard, 2360]
[Benjamin, Sisko,  2365]
```

```
Officers(
  firstname: string,
  lastname: string,
  id: int
)
```

What is a Relational Database?



Rows
(# = cardinality)

Columns
(# = degree/arity)

Specifies the name of the relation, plus the name and type of each column

The Data

```
[James,    Kirk,   2260]
[Jean Luc, Picard, 2360]
[Benjamin, Sisko,  2365]
```

```
Officers(
  firstname: string,
  lastname: string,
  id: int
)
```

You can think of a relation as a **set** of rows or **tuples**

Relation Example

Officers

FirstName,	LastName,	Ship
[James,	Kirk,	1701A]
[Leonard,	McCoy,	1701A]
[Spock,	SonOfSarek,	1701A]
[Montgomery,	Scott,	1701A]
[Hikaru,	Sulu,	2000]
[Pavel,	Chekov,	1701A]
[Nyota,	Uhura,	1701A]
[Christine,	Chapel,	0001]

Relation Example

Officers

Arity: 3

Cardinality: 9

FirstName,	LastName,	Ship
[James,	Kirk,	1701A]
[Leonard,	McCoy,	1701A]
[Spock,	SonOfSarek,	1701A]
[Montgomery,	Scott,	1701A]
[Hikaru,	Sulu,	2000]
[Pavel,	Chekov,	1701A]
[Nyota,	Uhura,	1701A]
[Christine,	Chapel,	0001]

Relational Query Languages

- The relational model supports simple, but powerful **querying** of data.
- Queries can be written intuitively.
- The database is responsible for efficient evaluation.
  - The query language has precise semantics
  - An optimizer can rewrite the query into a more efficient form without changing the semantics.

SQL

- Developed by IBM (for System R) in the 1970s.
- Standard used by many vendors.
  - SQL-86 (original standard)
  - SQL-89 (minor revisions; integrity constraints)
  - SQL-92 (major revision; basis for modern SQL)
  - SQL-99 (XML, window queries, generated default values)
  - SQL 2003 (major revisions to XML support)
  - SQL 2008 (minor extensions)
  - SQL 2011 (minor extensions; temporal databases)

A Basic SQL Query

```
SELECT   [DISTINCT] target-list

FROM     relation-list

WHERE    condition
```

A Basic SQL Query

```
SELECT   [DISTINCT] target-list

FROM     relation-list
```

A list of relation names

(possibly with a range-variable after each name)

```
WHERE    condition
```

A Basic SQL Query

```
SELECT   [DISTINCT] target-list
```

A list of attributes of relations in relation-list

```
FROM     relation-list
```

A list of relation names
(possibly with a range-variable after each name)

```
WHERE    condition
```

A Basic SQL Query

```
SELECT  [DISTINCT] target-list
```

A list of attributes of relations in relation-list

```
FROM    relation-list
```

A list of relation names
(possibly with a range-variable after each name)

```
WHERE   condition
```

Comparisons ('=', '<>', '<', '>', '<=', '>=') and other boolean predicates,
combined using AND, OR, and NOT
(a boolean formula)

A Basic SQL Query

(optional) keyword indicating that the answer should **not** contain duplicates

```
SELECT   [DISTINCT] target-list
```

A list of attributes of relations in relation-list

```
FROM     relation-list
```

A list of relation names
(possibly with a range-variable after each name)

```
WHERE    condition
```

Comparisons ('=', '<>', '<', '>', '<=', '>=') and other boolean predicates,
combined using AND, OR, and NOT
(a boolean formula)

Query Evaluation

```
SELECT   [DISTINCT] target-list
FROM     relation-list
WHERE    condition
```

1) Compute the 2^n combinations of tuples in all relations appearing in relation-list

2) Discard tuples that fail the condition

3) Delete attributes not in target-list

4) If DISTINCT is specified, eliminate duplicate rows

Query Evaluation

```
SELECT   [ UNIQUE ] target-list
FROM     relation-list
WHERE    condition
```

1) Compute the 2^n combinations of tuples in all relations appearing in relation-list

2) Discard tuples that fail the condition

3) Delete attributes not in target-list

4) If UNIQUE is specified, eliminate rows with duplicates

Query Evaluation

```
SELECT   [ UNIQUE ] target-list
FROM     relation-list
WHERE    condition
```

1) Compute the 2^n combinations of tuples in all relations appearing in relation-list

2) Discard tuples that fail the condition

3) Delete attributes not in target-list

4) If UNIQUE is specified, eliminate rows with duplicates

This is the least efficient strategy to compute a query!

A good optimizer will find more efficient strategies to compute **the same answer**.

Example-Condition

Find all officers on the Enterprise (Ship 1701A)

```
SELECT *
FROM Officers O
WHERE O.Ship = ‘1701A’
```

FirstName,	LastName,	Ship
[James,	Kirk,	1701A]
[Leonard,	McCoy,	1701A]
[Spock,	SonOfSarek,	1701A]
[Montgomery,	Scott,	1701A]
[Hikaru,	Sulu,	2000]
[Pavel,	Chekov,	1701A]
[Nyota,	Uhura,	1701A]
[Christine,	Chapel,	0001]

Example-Condition

Find all officers on the Enterprise (Ship 1701A)

```
SELECT *
FROM Officers O
WHERE O.Ship = '1701A'
```

FirstName,	LastName,	Ship
[James,	Kirk,	1701A]
[Leonard,	McCoy,	1701A]
[Spock,	SonOfSarek,	1701A]
[Montgomery,	Scott,	1701A]
[Pavel,	Chekov,	1701A]
[Nyota,	Uhura,	1701A]

FirstName,	LastName,	Ship
[James,	Kirk,	1701A]
[Leonard,	McCoy,	1701A]
[Spock,	SonOfSarek,	1701A]
[Montgomery,	Scott,	1701A]
[Hikaru,	Sulu,	2000]
[Pavel,	Chekov,	1701A]
[Nyota,	Uhura,	1701A]
[Christine,	Chapel,	0001]

Example-Condition

Find all officers on the Enterprise (Ship 1701A)

'*' denotes all attributes

'O.*' denotes all attributes in O

```
SELECT *
FROM Officers O
WHERE O.Ship = '1701A'
```

FirstName,	LastName,	Ship
[James,	Kirk,	1701A]
[Leonard,	McCoy,	1701A]
[Spock,	SonOfSarek,	1701A]
[Montgomery,	Scott,	1701A]
[Pavel,	Chekov,	1701A]
[Nyota,	Uhura,	1701A]

FirstName,	LastName,	Ship
[James,	Kirk,	1701A]
[Leonard,	McCoy,	1701A]
[Spock,	SonOfSarek,	1701A]
[Montgomery,	Scott,	1701A]
[Hikaru,	Sulu,	2000]
[Pavel,	Chekov,	1701A]
[Nyota,	Uhura,	1701A]
[Christine,	Chapel,	0001]

Example-Target List

Find just **names** of all officers on the Enterprise

```
SELECT O.FirstName,O.LastName
FROM Officers O
WHERE O.Ship = '1701A'
```

FirstName,	LastName,	Ship
[James,	Kirk,	1701A]
[Leonard,	McCoy,	1701A]
[Spock,	SonOfSarek,	1701A]
[Montgomery,	Scott,	1701A]
[Hikaru,	Sulu,	2000]
[Pavel,	Chekov,	1701A]
[Nyota,	Uhura,	1701A]
[Christine,	Chapel,	0001]

Example-Target List

Find just **names** of all officers on the Enterprise

```
SELECT O.FirstName,O.LastName
FROM Officers O
WHERE O.Ship = '1701A'
```

FirstName,	LastName
[James,	Kirk]
[Leonard,	McCoy]
[Spock,	SonOfSarek]
[Montgomery,	Scott]
[Pavel,	Chekov]
[Nyota,	Uhura]

FirstName,	LastName,	Ship
[James,	Kirk,	1701A]
[Leonard,	McCoy,	1701A]
[Spock,	SonOfSarek,	1701A]
[Montgomery,	Scott,	1701A]
[Hikaru,	Sulu,	2000]
[Pavel,	Chekov,	1701A]
[Nyota,	Uhura,	1701A]
[Christine,	Chapel,	0001]

Example-Multiple Relations

In English, what does this query compute?

```
SELECT O.FirstName,O.LastName
FROM Officers O, Ships S
WHERE O.Ship = S.ID
  AND S.Location = 'Vulcan'
```

FirstName,	LastName,	Ship
[James,	Kirk,	1701A]
[Leonard,	McCoy,	1701A]
[Spock,	SonOfSarek,	1701A]
[Montgomery,	Scott,	1701A]
[Hikaru,	Sulu,	2000]
[Pavel,	Chekov,	1701A]
[Nyota,	Uhura,	1701A]
[Christine,	Chapel,	0001]

ID,	Name,	Location
[1701A,	Enterprise-A,	Andoria]
[2000,	Excelsior,	Vulcan]
[1864,	Reliant,	Ceti Alpha VI]

Example-Multiple Relations

In English, what does this query compute?

```
SELECT O.FirstName,O.LastName
FROM Officers O, Ships S
WHERE O.Ship = S.ID
  AND S.Location = 'Vulcan'
```

FirstName,	LastName,	Ship
[James,	Kirk,	1701A]
[Leonard,	McCoy,	1701A]
[Spock,	SonOfSarek,	1701A]
[Montgomery,	Scott,	1701A]
[Hikaru,	Sulu,	2000]
[Pavel,	Chekov,	1701A]
[Nyota,	Uhura,	1701A]
[Christine,	Chapel,	0001]

ID,	Name,	Location
[1701A,	Enterprise-A,	Andoria]
[2000,	Excelsior,	Vulcan]
[1864,	Reliant,	Ceti Alpha VI]

Who is on a ship located at Vulcan?

Example-Multiple Relations

In English, what does this query compute?

```
SELECT O.FirstName,O.LastName
FROM Officers O, Ships S
WHERE O.Ship = S.ID
  AND S.Location = 'Vulcan'
```

↓ ↓ ↓

FirstName,	LastName
[Hikaru,	Sulu]

Who is on a ship located at Vulcan?

FirstName,	LastName,	Ship
[James,	Kirk,	1701A]
[Leonard,	McCoy,	1701A]
[Spock,	SonOfSarek,	1701A]
[Montgomery,	Scott,	1701A]
[Hikaru,	Sulu,	2000]
[Pavel,	Chekov,	1701A]
[Nyota,	Uhura,	1701A]
[Christine,	Chapel,	0001]

ID,	Name,	Location
[1701A,	Enterprise-A,	Andoria]
[2000,	Excelsior,	Vulcan]
[1864,	Reliant,	Ceti Alpha VI]

Range Variables

```
SELECT O.FirstName,O.LastName
FROM   Officers O, Ships S
WHERE  O.Ship = S.ID
  AND  S.Location = ‘Vulcan’
```

Range Variables

```
SELECT O.FirstName,O.LastName
FROM   Officers O, Ships S
WHERE  O.Ship = S.ID
  AND  S.Location = 'Vulcan'
```

is the same as

```
SELECT Officers.FirstName,Officers.LastName
FROM   Officers, Ships
WHERE  Officers.Ship = Ships.ID
  AND  Ships.Location = 'Vulcan'
```

Range Variables

```
SELECT O.FirstName,O.LastName
FROM   Officers O, Ships S
WHERE  O.Ship = S.ID
  AND  S.Location = 'Vulcan'
```

is the same as

```
SELECT Officers.FirstName,Officers.LastName
FROM   Officers, Ships
WHERE  Officers.Ship = Ships.ID
  AND  Ships.Location = 'Vulcan'
```

is the same as

```
SELECT FirstName,LastName
FROM   Officers, Ships
WHERE  Ship = ID
  AND  Location = 'Vulcan'
```

Range Variables

```
SELECT O.FirstName,O.LastName
FROM   Officers O, Ships S
WHERE  O.Ship = S.ID
  AND  S.Location = 'Vulcan'
```

is the same as

```
SELECT Officers.FirstName,Officers.LastName
FROM   Officers, Ships
WHERE  Officers.Ship = Ships.ID
  AND  Ships.Location = 'Vulcan'
```

is the same as

```
SELECT FirstName,LastName
FROM   Officers, Ships
WHERE  Ship = ID
  AND  Location = 'Vulcan'
```

But it's good style to use range variables and fully-qualified attribute names!

Expressions

```
SELECT O.age,
       age1 = O.age*0.2,
       O.age*3.0 AS age2
FROM   Officers O
```

↓

[age, age1, age2]

Arithmetic expressions can appear in targets or conditions.
Use '=' or 'AS' to assign names to these attributes.
(The behavior of unnamed attributes is unspecified)

Strings

```
SELECT O.FirstName, O.LastName
FROM   Officers O
WHERE  S.LastName LIKE ‘Ch%e%’
```

↓

```
[Pavel,      Chekov]
[Christine,  Chapel]
```

Sql uses single quotes for `‘string literals’`

Strings

```
SELECT O.FirstName, O.LastName
FROM   Officers O
WHERE  S.LastName LIKE ‘Ch%e%’
```

↓

```
[Pavel,      Chekov]
[Christine,  Chapel]
```

LIKE is used for String Matches

‘%’ matches 0 or more characters

(like RegEx `/.*/`)

Strings

```
SELECT O.FirstName, O.LastName
FROM   Officers O
WHERE  S.LastName LIKE 'Ch_%e%'
```

↓

~~[Pavel, Chekov]~~
[Christine, Chapel]

LIKE is used for String Matches

'_' matches exactly 1 character

(like RegEx /./)

UNION

Can be used to compute the *union* of any two **union-compatible** sets of tuples

```
SELECT O.FirstName
FROM Officers O
WHERE O.LastName = 'Kirk'
   OR O.LastName = 'Picard'
```

is the same as

```
SELECT O.FirstName FROM Officers O
WHERE O.LastName = 'Kirk'

UNION

SELECT O.FirstName FROM Officers O
WHERE O.LastName = 'Picard'
```

UNION

```
SELECT O.FirstName
FROM Officers O
WHERE O.LastName = 'Kirk'
   OR O.LastName = 'Picard'
```

What happens if we replace OR with AND?

UNION

```
SELECT O.FirstName FROM Officers O
WHERE O.LastName = 'Kirk'

EXCEPT

SELECT O.FirstName FROM Officers O
WHERE O.LastName = 'Picard'
```

Also available: `EXCEPT`

How would you expect that to work?

UNION

```
SELECT O.FirstName FROM Officers O
WHERE O.LastName = 'Kirk'

EXCEPT

SELECT O.FirstName FROM Officers O
WHERE O.LastName = 'Picard'
```

Also available: `EXCEPT`

How would you expect that to work?

Tuples in Set 1 that are NOT in Set 2

INTERSECT

Compute the *intersection* of two **union-compatible** sets of tuples

```
SELECT O.FirstName, O.LastName
FROM Officers O, Visited V
WHERE O.ID = V.Officer
  AND V.Planet = 'Vulcan'

INTERSECT

SELECT O.FirstName, O.LastName
FROM Officers O, Visited V
WHERE O.ID = V.Officer
  AND V.Planet = 'Andoria'
```

Part of SQL/92, but not universally supported.

How would you rewrite this query?

Nested Queries

What does this query compute?

```
SELECT O.FirstName, O.LastName
FROM Officers O
WHERE O.ID IN (SELECT V.Officer
               FROM   Visited V
               WHERE  V.Planet = 'Vulcan')
```

Nested Queries

What does this query compute?

```
SELECT O.FirstName, O.LastName
FROM Officers O
WHERE O.ID IN (SELECT V.Officer
               FROM   Visited V
               WHERE  V.Planet = 'Vulcan')
```

IN nested query must have exactly **one** attribute

Nested Queries

What does this query compute?

```
SELECT O.FirstName, O.LastName
FROM Officers O
WHERE O.ID IN (SELECT V.Officer
               FROM   Visited V
               WHERE  V.Planet = 'Vulcan')
```

Use `NOT IN` for all officers who have never visited 'Vulcan'

`IN` nested query must have exactly **one** attribute

Nested Queries

(With Correlation)

```
SELECT O.FirstName, O.LastName
FROM Officers O
WHERE EXISTS (SELECT *
              FROM   Visited V
              WHERE  V.Planet = 'Vulcan'
                AND  O.ID = V.Officer)
```

`EXISTS` is true if the nested query returns at least one result

The nested query can refer to attributes from the outer query

Nested Queries

(With Correlation)

```
SELECT O.FirstName, O.LastName
FROM Officers O
WHERE EXISTS (SELECT *
              FROM   Visited V
              WHERE  V.Planet = 'Vulcan'
                AND  O.ID = V.Officer)
```

`EXISTS` is true if the nested query returns at least one result

The nested query can refer to attributes from the outer query

Nested Queries

(With Correlation)

```
SELECT O.FirstName, O.LastName
FROM Officers O
WHERE EXISTS (SELECT UNIQUE V.Officer
              FROM   Visited V
              WHERE  O.ID = V.Officer)
```

What does this query ask?

Nested Queries

(With Correlation)

```
SELECT O.FirstName, O.LastName
FROM Officers O
WHERE EXISTS (SELECT UNIQUE V.Officer
              FROM   Visited V
              WHERE  O.ID = V.Officer)
```

What does this query ask?

Why V.Officer and not *?

More Set Operators

```
IN

EXISTS

UNIQUE
```

More Set Operators

IN → NOT IN

EXISTS → NOT EXISTS

UNIQUE → NOT UNIQUE

More Set Operators

[op] ANY [op] ALL

```
SELECT * FROM Officers O
WHERE O.Rank > ALL (SELECT O2.rank
                    FROM Officers O2,
                         Ships S
                    WHERE O2.Ship = S.ID
                      AND S.Name = 'Enterprise'
                    )
```

What does this compute?

From-Nesting

```
SELECT *
FROM Officers O,
     (SELECT V.Officer
      FROM Visited V
      WHERE V.Planet = 'Andoria'
     ) A
WHERE O.ID = A.Officer
```

Queries are relations!

From-Nesting

```
SELECT *
FROM Officers O,
     (SELECT V.Officer
      FROM Visited V
      WHERE V.Planet = 'Andoria'
     ) A
WHERE O.ID = A.Officer
```

Queries are relations!

Aggregate Operators

```
SELECT COUNT(*)
FROM Officers O, Ships S
WHERE O.Ship = S.ID
  AND S.Name = 'Enterprise'
```

What does this compute?

Aggregate Operators

```
COUNT(*)
COUNT(DISTINCT A[, B[, …]])
SUM([DISTINCT] A)
AVG([DISTINCT] A)
MAX(A)
MIN(A)
```

Single Column/Expression

Aggregate Operators

```
SELECT * FROM Officers O
WHERE O.Rank > ALL (SELECT O2.rank
                    FROM Officers O2,
                         Ships S
                    WHERE O2.Ship = S.ID
                      AND S.Name = 'Enterprise'
                    )
```

How could you write this query without ALL?

Aggregate Operators

```
SELECT S.Name, AVG(O.Age)
FROM Officers O, Ships S
WHERE O.Ship = S.ID
```

Aggregate Operators

This query is illegal!

Why?

```
SELECT S.Name, AVG(O.Age)
FROM Officers O, Ships S
WHERE O.Ship = S.ID
```

Aggregate Operators

This query is illegal!

Why?

```
SELECT S.Name, AVG(O.Age)
FROM Officers O, Ships S
WHERE O.Ship = S.ID
```

Can't combine Aggregate and Non-Aggregate targets!

Aggregate Operators

This query is illegal!

Why?

```
SELECT S.Name, AVG(O.Age)
FROM Officers O, Ships S
WHERE O.Ship = S.ID
GROUP BY S.Name
```

Grouping allows us to apply aggregates to Groups of tuples.

Group-By Queries

```
SELECT [DISTINCT] target-list
FROM relation-list
WHERE condition
GROUP BY grouping-list
HAVING group-condition
```

The target-list contains
a) grouped attributes
b) aggregate expressions

The targets of type (a) must be a **subset** of the grouping-list

(intuitively each answer tuple corresponds to a single group, and each group must have a single value for each attribute)

Group-By Queries

```
SELECT [DISTINCT] target-list
FROM relation-list
WHERE condition
GROUP BY grouping-list
HAVING group-condition
```

The condition is applied before grouping
The having-condition is applied after grouping

How can we compute the Top 5 officers by rank?

```
SELECT O.Name, O.Rank
FROM Officers O
```

How can we compute the Top 5 officers by rank?

```
SELECT O.Name, O.Rank
FROM Officers O
ORDER BY O.Rank
```

Order By/Limit

How can we compute the Top 5 officers by rank?

```
SELECT O.Name, O.Rank
FROM Officers O
ORDER BY O.Rank
LIMIT 5
```

NULL Values

- Field values can be *unknown* or *inapplicable*.
  - An officer not assigned to a ship.
  - A aliens that have no last names.
    - ‘Spock’ or ‘Data’
- SQL provides a special NULL value for this.
- NULL makes things more complicated.

NULL Values

```
O.Rank > 3.0
```

What happens if O.Rank is NULL?

NULL Values

```
O.Rank > 3.0
```

What happens if O.Rank is NULL?

Predicates can be True, False, or Unknown (3-valued logic)

`WHERE` clause eliminates all **Non-True** values

NULL Values

```
O.Rank > 3.0
```

What happens if O.Rank is NULL?

Predicates can be True, False, or Unknown (3-valued logic)

`WHERE` clause eliminates all **Non-True** values

How does this interact with AND, OR, NOT?

Creating Relations in SQL

```
CREATE TABLE Officers
  ( FirstName CHAR(20),
    LastName  CHAR(20),
    Ship      CHAR(5),
    ID        INTEGER
  )

CREATE TABLE Ships
  ( ID        CHAR(5),
    Name      CHAR(20),
    Location  CHAR(40)
  )
```

The schema defines not only the column names, but also their types (domains)

Creating Relations in SQL

```
CREATE TABLE Officers
  ( FirstName CHAR(20),
    LastName  CHAR(20),
    Ship      CHAR(5),
    ID        INTEGER
  )

CREATE TABLE Ships
  ( ID        CHAR(5),
    Name      CHAR(20),
    Location  CHAR(40)
  )
```

The schema defines not only the column names, but also their types (domains)

For example a 20-character string

Modifying Relations

Destroy the relation 'Officers'
All schema information AND tuples are deleted

```
DROP TABLE Officers
```

Add a new column (field) to the Ships relation
Every tuple in the current instance is extended with a 'null' value in the new field

```
ALTER TABLE Ships
  ADD COLUMN Commissioned DATE
```

Adding and Deleting Tuples

Insert single tuples using:

```
INSERT INTO Officers (FirstName, LastName, Ship)
  VALUES ('Benjamin', 'Sisko', '74205')
```

Can delete all tuples satisfying some condition (e.g., Ship = 2000)

```
DELETE FROM Officers O
  WHERE O.Ship = '2000'
```

More powerful data manipulation commands are available in SQL
(We'll discuss them later in the course)

Group Question

```
Officers(
  Name Char(100),
  ID Int,
  YearsOfService Float,
)
```

Find the name of all officers who are above the mean number of years of service.

Find 2-3 people around you and write this query down.

Summary

- Relations have a schema, rows, and columns
- SQL is a language for querying relations
  - `SELECT` to access (query) data
    - Different features for different access patterns.
  - `INSERT INTO`, `DELETE FROM` to modify data
  - `CREATE TABLE`, `DROP TABLE`, `ALTER TABLE` to modify relations
- `NULL` complicates things