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

(optional) keyword indicating that the answer should **not** contain duplicates SELECT [DISTINCT] target-list A list of attributes of relations in relationrelation-list FROM A list of relation names (possibly with a range-variable after each name) condition WHERE

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

A Basic SQL Query

net.sf.jsqlparser.statement.select.PlainSelect

SELECT [DISTINCT] target-list
mySelect.getDistinct() mySelect.getSelectItems()

FROM relation-list
mySelect.getFromItem() and mySelect.getJoins()

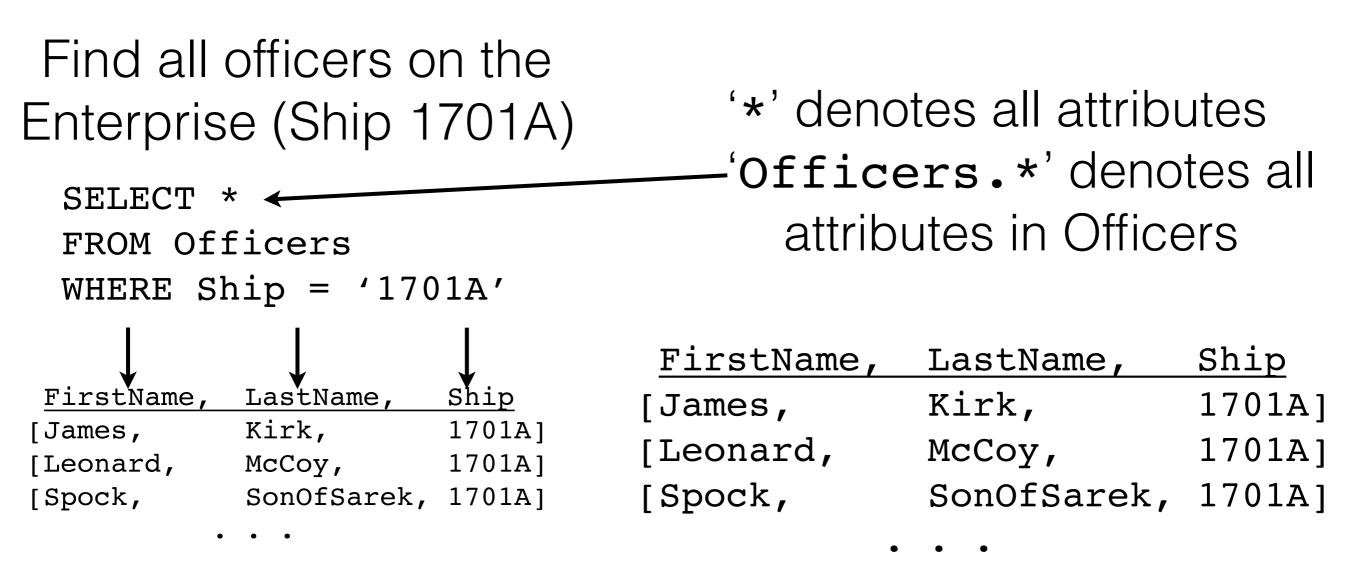
WHERE condition

Query Evaluation

- SELECT [DISTINCT] target-list FROM relation-list WHERE condition
- 1) Compute the 2ⁿ 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

This is the least efficient strategy to compute a query! A good optimizer will find more efficient strategies to compute **the same answer**.

Example-Wildcards



net.sf.jsqlparser.statement.select.AllColumns
net.sf.jsqlparser.statement.select.AllTableColumns

Example-Condition

Find all officers on the Enterprise (Ship 1701A)

| SELECT * | | | |
|----------------------|--------------|--------------|--|
| FROM Off | icers | | |
| WHERE Ship = '1701A' | | | |
| | \checkmark | \downarrow | |
| <u>FirstName</u> , | 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

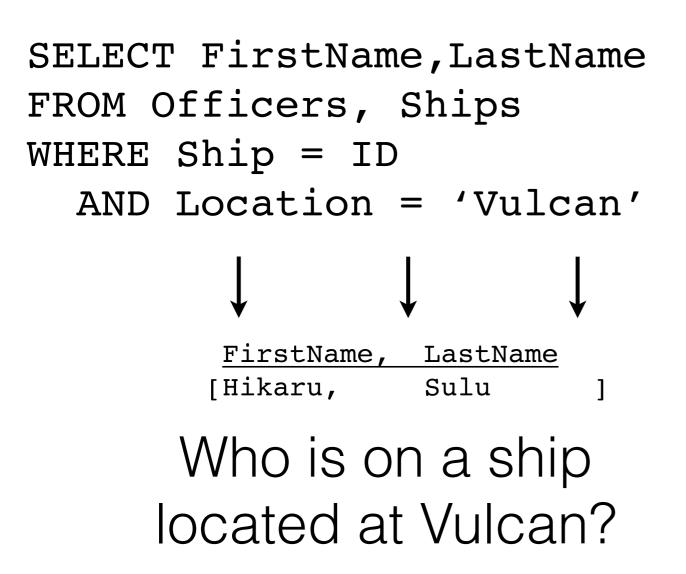
```
SELECT O.FirstName,O.LastName
FROM Officers O
WHERE O.Ship = '1701A'
```

| <u>FirstName,</u> | LastName | |
|-------------------|-----------|-----|
| [James, | Kirk |] |
| [Leonard, | МсСоу |] |
| [Spock, | SonOfSare | k] |
| [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?



| 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 FirstName, LastName

FROM Officers, Ships

WHERE Ship = ID

AND Location = 'Vulcan'

is the same as

But it's good style to use range variables and fullyqualified attribute names!

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

is the same as

JSqlParser calls this an "alias"

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

Expressions

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

SQL uses single quotes for 'string literals'

Strings

SELECT O.FirstName, O.LastName FROM Officers O WHERE O.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 O.LastName LIKE 'Ch %e%' Favel, Chekov] [Christine, Chapel] **LIKE** is used for String Matches '%' matches 0 or more characters (like RegEx / . */)

UNION

Computes 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

net.sf.jsqlparser.statement.select.Union

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

myUnion.getPlainSelects()

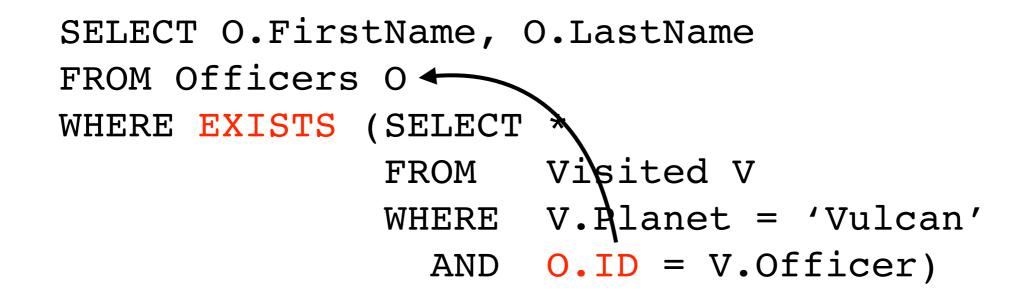
Nested Queries

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'

net.sf.jsqlparser.expression.operators.relational.InExpression
 net.sf.jsqlparser.statement.select.SubSelect

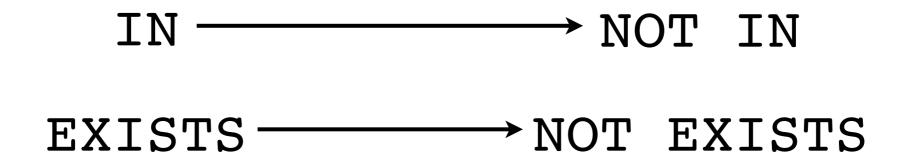




EXISTS is true if the nested query returns at least one result The nested query can refer to attributes from the outer query

net.sf.jsqlparser.expression.operators.relational.ExistsExpression

More Set Operators



More Set Operators

[op] ANY [op] ALL

SELECT * FROM Officers O WHERE O.Rank > ALL (SELECT O2.rank FROM Officers 02, Ships S WHERE O2.Ship = S.IDAND S.Name = 'Enterprise' What does this compute? Which officers outrank every officer on the Enterprise? net.sf.jsqlparser.expression.AllComparisonExpression

From-Nesting

Queries are relations!

net.sf.jsqlparser.statement.select.SubSelect

Aggregate Operators

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

What does this compute? How many officers are on the Enterprise?

net.sf.jsqlparser.expression.Function

```
COUNT(*)
COUNT(DISTINCT A[, B[, ...]])
SUM([DISTINCT] A)
AVG([DISTINCT] A)
MAX ( A
           Single Column/Expression
MIN(A)
```

Aggregate Operators

Group Exercise

How could you write this query without ALL?

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 athoring using any alterange to the start of the

Group-By Queries

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

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

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

Group-By Queries

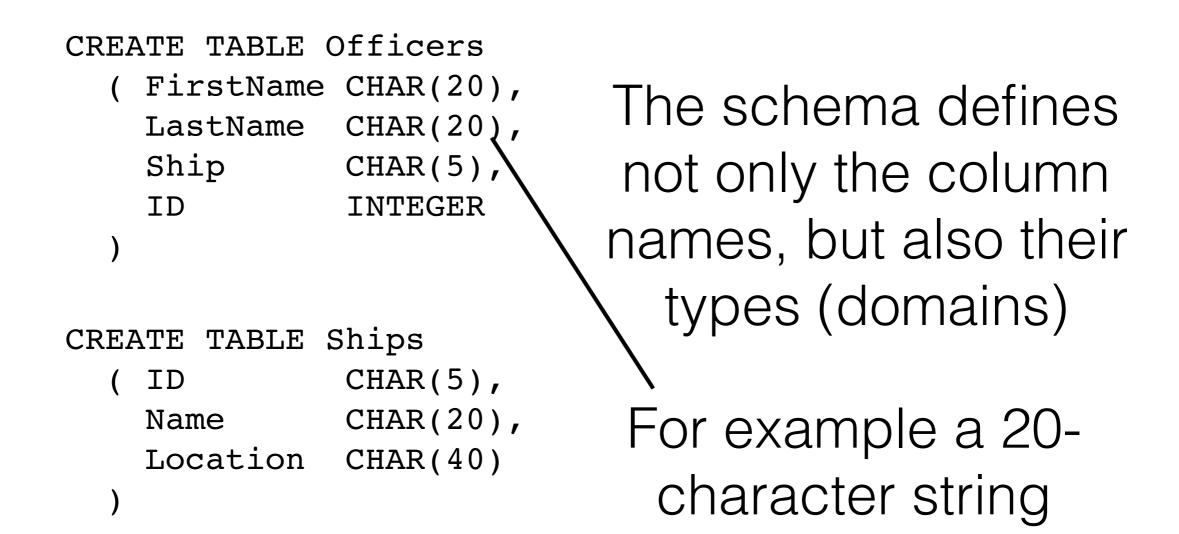
SELECT [DISTINCT] target-list FROM relation-list WHERE condition GROUP BY grouping-list HAVING group-condition mySelect.getHaving()

mySelect.getGroupByColumnReferences()

Order By/Limit

How can we compute the Top 5 officers by rank?

Defining Relations in SQL



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)

SQL

- 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
- Next time...
 - Translating SQL to Relational Algebra (equivalence)