Project Outline

SQL Query → Parser & Translator → Relational Algebra

JSqlParser

Query Result → Hope and Duct Tape?
SQL is Human Readable

• Lots of Syntactic Sugar
  • WHERE vs HAVING

• Lots of Corner Cases
  • SELECT A, B vs SELECT A, SUM(B)

• Non-obvious evaluation strategy
  • SELECT … FROM R, S, T, … WHERE …

SQL is hard to evaluate directly!
Relational Algebra

• Equivalent to SQL (to be discussed)
• SIMPLE! (only a handful of operators)
• “Non-declarative” (easy to rewrite)
• Minimal corner cases or syntactic sugar

“RA” is easier to interpret!
Relational Algebra

• Basic Relational Operators
  • Select (σ), Project (π), Cross/Join (⋈), Union (U), Relation (R, S, T, ...), Minus (-)

• Extended Relational Operators (more next week)
  • Aggregates (SUM, COUNT, MIN/MAX, AVERAGE)

• List Operators: Sort, Limit
The Evaluation Pipeline

1. Parsed Query
2. How does this work? (later today)
3. Data
4. What does this look like? (today)
5. Results
6. How does this work? (next class)
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

A list of attributes of relations in relation-list

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

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

(keyword indicating that the answer should not contain duplicates)

SELECT [DISTINCT] target-list

FROM relation-list

WHERE condition
A Basic SQL Query

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

```java
mySelect.getDistinct()
mySelect.getSelectItems()

mySelect.getFromItem()
and mySelect.getJoins()

mySelect.getWhere()
```
Query Evaluation

```sql
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

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

Why do you explicitly indicate that you want duplicate elimination in SQL?
Example-Wildcards

Find all officers on the Enterprise (Ship 1701A)

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

‘*’ denotes all attributes
‘Officers.*’ denotes all attributes in Officers

<table>
<thead>
<tr>
<th>FirstName</th>
<th>LastName</th>
<th>Ship</th>
</tr>
</thead>
<tbody>
<tr>
<td>James</td>
<td>Kirk</td>
<td>1701A</td>
</tr>
<tr>
<td>Leonard</td>
<td>McCoy</td>
<td>1701A</td>
</tr>
<tr>
<td>Spock</td>
<td>SonOfSarek</td>
<td>1701A</td>
</tr>
</tbody>
</table>

...
Example-Condition

Find all officers on the Enterprise (Ship 1701A)

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

<table>
<thead>
<tr>
<th>FirstName</th>
<th>LastName</th>
<th>Ship</th>
</tr>
</thead>
<tbody>
<tr>
<td>James</td>
<td>Kirk</td>
<td>1701A</td>
</tr>
<tr>
<td>Leonard</td>
<td>McCoy</td>
<td>1701A</td>
</tr>
<tr>
<td>Spock</td>
<td>SonOfSarek</td>
<td>1701A</td>
</tr>
<tr>
<td>Montgomery</td>
<td>Scott</td>
<td>1701A</td>
</tr>
<tr>
<td>Hikaru</td>
<td>Sulu</td>
<td>2000</td>
</tr>
<tr>
<td>Pavel</td>
<td>Chekov</td>
<td>1701A</td>
</tr>
<tr>
<td>Nyota</td>
<td>Uhura</td>
<td>1701A</td>
</tr>
<tr>
<td>Christine</td>
<td>Chapel</td>
<td>0001</td>
</tr>
</tbody>
</table>
Example-Target List

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

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

[James, Kirk      ]
[Leonard, McCoy   ]
[Spock, SonOfSarek]
[Montgomery, Scott]
[Hikaru, Sulu     ]
[Pavel, Chekov    ]
[Nyota, Uhura     ]

[Christine, Chapel, 0001 ]
```

```sql
SELECT O.FirstName, O.LastName, O.Ship
FROM Officers O
WHERE O.Ship = '1701A'
```
Example-Multiple Relations

In English, what does this query compute?

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

Who is on a ship located at Vulcan?
Example-Multiple Relations

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

mySelect\texttt{.getFromItem()} returns
\ldots\texttt{.schema.Table(Officers)}

mySelect\texttt{.getJoins()} returns
\texttt{List(}
\ldots\texttt{.select.Join(Table(Ships), \{simple\})} )
Range Variables

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

JetSqlParser calls this an “alias”

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

SELECT 0.age,
       age1 = 0.age*0.2,
       0.age*3.0 AS age2
FROM Officers 0

[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  O.LastName LIKE 'Ch%e%'
```

\[ \text{[Pavel, Chekov]} \]
\[ \text{[Christine, Chapel]} \]

**LIKE** is used for String Matches

‘%’ matches 0 or more characters

(like RegEx /.*/)
SELECT O.FirstName, O.LastName
FROM Officers O
WHERE O.LastName LIKE ‘Ch_%e%’

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’

IN nested query must have exactly one attribute

Use net.sf.jsqlparser.statement.select.SubSelect
net.sf.jsqlparser.expression.operators.relational.InExpression
net.sf.jsqlparser.statement.select.SubSelect
Nested Queries
(With Correlation)

```sql
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.

```java
net.sf.jsqlparser.expression.operators.relational.ExistsExpression
```
More Set Operators

IN ——— NOT IN

EXISTS ——— NOT EXISTS
More Set Operators

[\text{op}] \text{ANY} \quad [\text{op}] \text{ALL}

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

What does this compute?
Which officers outrank every officer on the Enterprise?

\text{net.sf.jsqlparser.expression.AllComparisonExpression}
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!

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?
Aggregate Operators

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

Single Column/Expression
Group Exercise

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

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
```

Why?

Can't combine Aggregate and Non-Aggregate targets!
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.getGroupByColumnReferences()

mySelect.getHaving()
Order By/Limit

How can we compute the Top 5 officers by rank?

```sql
SELECT O.Name, O.Rank
FROM Officers O
ORDER BY O.Rank
LIMIT 5
mySelect.getOrderByElements()
mySelect.getLimit()
```
Defining 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

\texttt{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

\texttt{ALTER TABLE Ships}
\texttt{ADD COLUMN Commissioned DATE}
Adding and Deleting Tuples

Insert single tuples using:

\[
\text{INSERT INTO Officers (FirstName, LastName, Ship) VALUES ('Benjamin', 'Sisko', '74205')}
\]

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

\[
\text{DELETE FROM Officers O WHERE O.Ship = '2000'}
\]

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)