Background (1 / 2)

- IBM’s System R was released in 1978
  - Its query language name: SEQUEL (Structured English QUEry Language)
    - But trademarked by British airplane company!
    - After dropping the vowels: SQL
- IBM’s current DB/2 was released in 1982; also used SQL
- SQL:
  - A marriage of TRC to RA
  - SQL = DML + DDL + DCL + QL
Background (2 / 2)

- SQL is no longer a proprietary language:
  - SQL is now an ANSI/ISO standard
- But no one strictly follows any of them!
  - There is a basic subset you can count on
  - Example: Tuple IDs are non-standard

Relational Operators (1 / 5)

But first: SQL’s SELECT statement

Example(s):
Relational Operators (2 / 5)

Now that we can perform $\pi$, we can answer our first standard query:

“What is the content of the Employee relation?”

Relational Operators (3 / 5)

Performing $\sigma$ requires a new clause:

Example(s):
These are also all of the clauses that we need for ⊲ ⊳:

Example(s):

For completeness, our fourth standard query:

Example(s):
Renaming Attributes

You may give your result relations new attribute names:

Example(s):

A Note about Duplicate Tuples

By default, SQL does not remove duplicate tuples from result relations.

But we can override that behavior!

Example(s):
Ordering Result Tuples

We can sort tuples, too, with the ORDER BY clause.

Example(s):

Computed Columns

We can perform basic arithmetic with field values:

Example(s):
Tuple Variables (a.k.a. Aliases)

We can assign relations temporary, alternate names.

Example(s):

Regular Expressions (1 / 2)

SQL allows us to search for values that match a particular pattern.

Form:
Regular Expressions (2 / 2)

Example(s):

Set Operators (1 / 5)

Cartesian Product (×)

Example(s):
Set Operators (2 / 5)

Union (∪)

Example(s):

Set Operators (3 / 5)

Difference (−) and Intersection (∩)

Example(s):
The Return of . . . Division!

Version #1: Relational Algebra expression

Recall: $\alpha \div \beta = \pi_{A-B}(\alpha) - \pi_{A-B}((\pi_{A-B}(\alpha) \times \beta) - \alpha)$

And our sample division query:

“Find the S#s of the suppliers who supply all parts of weight equal to 17.”

And so, $\alpha \div \beta = \pi_{A-B}(\alpha) - \pi_{A-B}((\pi_{A-B}(\alpha) \times \beta) - \alpha)$ becomes in SQL:

```sql
select distinct sno from spj
except
select sno from
  ( select sno, pno
      from (select sno from spj) as t1,
           (select pno from p where weight=17) as t2
      except
       select sno, pno from spj
  ) as t3;
```
Aggregate Functions (1 / 3): Background

Idea: Let SQL compute basic statistical results for us

SQL provides aggregate functions for this purpose:

- `count([distinct] attr)` — counting tuples in a relation
- `sum([distinct] attr)` — totaling values of `attr` in a relation
- `avg([distinct] attr)` — averaging values of `attr` in a relation
- `min(attr)` — smallest value of `attr` in a relation
- `max(attr)` — largest value of `attr` in a relation

Aggregate Functions (2 / 3)

Example(s):
Aggregate Functions (3 / 3)

Example(s):

If we have one of each part in a box, how much does the content weigh?

Group By

Example(s):

What are the average quantities in which suppliers are supplying parts?
Having

Example(s):
Which suppliers are supplying parts in average quantity under 400, and what are those averages?

More on Nested Queries (1 / 4)
We've seen this idea before (e.g., the division query)

Example(s):
Remember this query?
Example(s):

Idea: Create a set of parts available in quantity > 200, and test each part from the DB against that set.

To create the P#s of the ‘quantity > 200’ parts:

```sql
select pno
from spj
where qty > 200;
```

And to produce the names of the parts in that set:

More on Nested Queries (3 / 4)

Notes:
More on Nested Queries (4 / 4)

One more operator: EXISTS

Example(s):

Another (awkward!) version of the qty > 200 query:

Division, Revisited (1 / 6)

Version #2: “Double ∃”

Consider: “Find the S#s of the suppliers who supply all parts of weight equal to 17.”
“Find S#s such that ∃ parts of weight 17 for which ∃ suppliers that supply them all” in SQL:

```sql
select distinct sno
from   spj as global
where  not exists
   ( select pno
    from   p
    where  weight = 17 and not exists
       ( select *
        from   spj as local
        where  local.pno = p.pno
            and local.sno = global.sno
       )
   )
```

Version #3: Set Containment
```sql
select distinct sno
from spj as global
where not exists (  
  ( select pno
    from p  
    where weight = 17
  ) except (  
    select p.pno
    from p, spj
    where p.pno = spj.pno
    and spj.sno = global.sno
  )
)
```

Version #4: Set Cardinality
Division, Revisited (6 / 6)

```sql
select distinct sno
from spj, p
where spj.pno = p.pno and weight = 17
having count(distinct p.pno) =
    ( select count (distinct pno)
        from p
        where weight = 17
    )
```

Outer Joins (1 / 5)

Regular (“inner”) joins discard non-matching tuples.

**Example(s):**

<table>
<thead>
<tr>
<th>M</th>
<th>Id</th>
<th>Name</th>
<th>N</th>
<th>Building</th>
<th>Supervisor</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>Roy</td>
<td></td>
<td>A</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Amy</td>
<td></td>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Joy</td>
<td></td>
<td>C</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>D</td>
<td>NULL</td>
</tr>
</tbody>
</table>
Now consider this slightly different query.

Example(s):

Three varieties of outer join:

- Left Outer Join (◯⃝): Retains unmatched tuples from left relation
- Right Outer Join (⃝◯): Retains unmatched tuples from right relation
- Full Outer Join (◯⃝): Retains all unmatched tuples
Outer Joins (4 / 5)

The SQL outer join syntax:

```
select <attribute list>
from ( <relation> [left/right/full] outer join <relation> on <join condition> )
where <condition> ;
```

Example(s):

```
```

Outer Joins (5 / 5)

Outer join is not an fundamental operator.

Example(s):

```
Name all employees and the buildings they supervise.
```

```
```
SQL as DDL

First order of business: Creating a database!

The exact mechanism depends on the DBMS.

1. Postgres:  $ createdb <name>
2. Oracle:  CREATE DATABASE <name>;

Creating Relations (1 / 3)

First: Know your available attribute types!

Examples:

- **Integers**: integer, number(p)
- **Floats**: float, real, number(p,s)
- **Strings**: char(n), varchar(n), varchar2(n)
- **Others**: timestamp, blob, bfile,...
Creating Relations (2 / 3)

To create a relation:

```
CREATE TABLE <table name> (  
    <attribute name> <data type> [ NOT NULL ],
    . . .
    [ PRIMARY KEY (<attribute>) ]
)
```

Creating Relations (3 / 3)

Example(s):

```
Creating the supplier (S) relation:
create table s (  
    sno varchar2(5), -- the supplier ID number
    sname varchar2(20), -- the supplier’s name
    status integer, -- supplier status
    city varchar2(15), -- location of supplier
    primary key (sno)
)
```

SQL – CSc 460 v1.0 (McCann) – p. 43
Creating Indices (1 / 3)

Form:

```
CREATE [ UNIQUE ] INDEX <index name>
ON <table name>
[ USING <access method> ]
( <attribute name> [, <attribute name> ... ]);
```

Creating Indices (2 / 3)

Example(s):

Create an index on jno in SPJ:
Creating Views (1 / 2)

Recall: ANSI/SPARC External Layer

Form:

```
CREATE VIEW <view name> [ ( <attribute list> ) ]
    AS <select query>;
```
Creating Views (2 / 2)

Example(s):

Create a view of supplier names and the IDs of the parts that they supply.

View Updates (1 / 2)

Can we allow updates to views?

Example(s):
Example(s): (continued!)

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>y</td>
<td>1</td>
<td>a</td>
<td>6</td>
</tr>
<tr>
<td>y</td>
<td>1</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>2</td>
<td>c</td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>x</td>
<td>2</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>3</td>
<td>c</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>1</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>2</td>
<td>c</td>
<td></td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>d</td>
</tr>
<tr>
<td>6</td>
<td>y</td>
</tr>
<tr>
<td>1</td>
<td>y</td>
</tr>
<tr>
<td>7</td>
<td>x</td>
</tr>
</tbody>
</table>

SQL as DML
Inserting Tuples into a Relation

To insert a tuple into a relation:

```
INSERT INTO <relation name> [ ( <column list> ) ]
VALUES ( <expression list> );
```

Example(s):

---

Bulk Loading a Database

Example(s):

---
Updating Content of Tuples

To modify data in existing tuples:

**UPDATE** `<relation name>`

**SET** `<attribute name> = <expression> [,, . . . ]`

[ FROM `<relation list>` ]

[ WHERE `<condition>` ];

Example(s):

---

Deleting Tuples

Like updating, a condition is used to ID tuples for removal:

**DELETE FROM** `<relation name>`

**WHERE** `<condition>`;
Deleting Relations

To remove tables, indices, views, . . .

```
DROP { TABLE | INDEX | VIEW | DATABASE } <name>;
```
Wait! What About “SQL as DCL?”

We’ll cover that in **Topic 14: Security**.