

# Joins

CSC 256, SQL Programming

# Joins

- The basis of the join operation is to link a foreign key in one table to the primary key in another table, that is, the operands to the join operation are tables.
- Types of joins:
  - Cross join
  - Inner join
  - Outer join

# Cross Joins

- The cross join is the Cartesian product of the rows in both tables.
- Basic syntax:

```
select *  
from table1  
    cross join table2;
```

- The resulting number of rows is equal to the number of rows in the first table multiplied by the number of rows in the second table
- Conceptually, the cross join is the base for the other joins

# Cartesian Product

- Mathematical Definition: The *Cartesian Product* of two sets  $A$  and  $B$ , denoted by  $A \times B$  is the set of ordered pairs  $(a, b)$  where  $a \in A$  and  $b \in B$
- Example: Let  $A = \{a, b\}$  and  $B = \{1, 2, 3\}$ , then  $A \times B = \{(a, 1), (a, 2), (a, 3), (b, 1), (b, 2), (b, 3)\}$

# Cross Join Example

- T1

a1	a2
1	A
2	B

- T2

b1	b2
5	C
7	D

# Cross Join Example (continued)

■  $T1 \times T2$

a1	a2	b1	b2
1	A	5	C
1	A	7	D
2	B	5	C
2	B	7	D

■ 4 rows = 2 rows  $\times$  2 rows

# Cross Join Examples (Chinook)

```
select "EmployeeId", "CustomerId"  
from "Employee"  
      cross join "Customer";
```

EmployeeId		CustomerId
1		1
2		1
3		1
...		

(472 rows)

# Cross Join Examples (Chinook)

```
select "FirstName"  
from "Employee"  
    cross join "Customer";
```

ERROR: column reference "FirstName" is ambiguous

LINE 1: select "FirstName" from "Employee" cross join "Cust



# Qualifying Column Names

- When we start using joins, we often need to qualify column names to avoid ambiguity when both tables have a column with the same name
- The syntax to qualify a column name is `table.column`
- Example:

```
select "Employee"."FirstName"  
from "Employee"  
      cross join "Customer";
```

- When using joins it is considered a good practice to always qualify column names

# Qualifying Column Names (continued)

- When qualifying column names, we can alias the table names in the FROM clause with the AS keyword

- Example:

```
select e."FirstName"  
from "Employee" as e  
cross join "Customer";
```

- When selecting columns we can use the \* per qualified table name

- Example:

```
select e.*  
from "Employee" as e  
cross join "Customer";
```

# Old Cross Join Syntax

- There is an older syntax for cross joins where a comma is used as the cross join operator in the FROM clause:

```
select t1.*, t2.*  
from t1, t2;
```

- We should use the modern cross join syntax to avoid potential confusion.

# Inner Joins

- Basic syntax:

```
select *  
from table1  
    inner join table 2  
    on <condition>
```

- Conceptually, an inner join is a cross join followed by a step to remove rows that do not satisfy some property
- The inner join is useful because we typically have inter-table relationships with primary and foreign keys
- Example

```
select "TrackId", "Name", "Title"  
from "Track"  
    inner join "Album"  
    on "Track"."AlbumId" = "Album"."AlbumId";
```

# Inner Joins (continued)

- Sometimes we might need data from multiple tables so we might need more than one join operation
- Example

```
select
    "TrackId",
    t."Name" as "Track",
    al."Title" as "Album",
    ar."Name" as "Artist"
from "Track" as t
    inner join "Album" as al
        on t."AlbumId" = al."AlbumId"
    inner join "Artist" as ar
        on al."ArtistId" = ar."ArtistId";
```

# Inner Joins (continued)

- The keyword `join` implicitly means inner join
- Equality joins can be expressed with the `using` syntax if the column has the same name in both tables
- Example

```
select
    "TrackId",
    t."Name" as "Track",
    al."Title" as "Album",
    ar."Name" as "Artist"
from "Track" as t
    join "Album" as al using ("AlbumId")
    join "Artist" as ar using ("ArtistId");
```

# Old Inner Join Syntax

- There is an older inner join syntax that follows the older cross join syntax where the inner join condition is specified as a WHERE clause

```
select *  
from t1, t2  
where t1.a_id = t2.a_id;
```

- We should use the modern cross join syntax to avoid potential confusion.

# Outer Joins

- Joins (conceptually)
  - cross join: Cartesian product
  - inner join: cross join *plus* filtering step
  - outer join: cross join *plus* filtering step *plus* add in missing rows
- Outer join types:
  - left
  - right
  - full



# Outer Join Syntax

- Basic outer join syntax:

```
select *  
from t1  
      (left|right|full) [outer] join t2  
      on condition
```

- Note the outer keyword is implicit, so you *can* leave it out

# Outer Join Example

- Some artists do not have any albums

```
select
    "Artist"."ArtistId",
    "AlbumId"
from "Artist"
    left outer join "Album"
        on "Artist"."ArtistId" = "Album"."ArtistId"
where "AlbumId" is null;
```

# Outer Join Example (continued)

- Conceptual steps of the previous example
  - Cartesian product of both tables
  - filter out rows where the “ArtistId” values are not equal
  - add in missing rows from the “Artist” table
    - the rows that are added in have NULL values for the missing columns
    - the rows are added from the left table because we used a left outer join
- Note the right outer join would add in missing rows from the right table and the full outer join would add in missing rows from both tables

# Another Outer Join Example

- How many composers are on each album?
- First attempt:

```
select
    al."AlbumId",
    al."Title",
    count(*) as "Number of Artists"
from "Artist" as ar
    left outer join "Album" as al
        on ar."ArtistId" = al."ArtistId"
group by al."AlbumId", al."Title"
order by al."AlbumId";
```

# Another Outer Join Example

- How many composers are on each album?
- We need to consider that aggregate functions disregard nulls and the left outer join adds in null values:

```
select
    a."AlbumId",
    a."Title",
    count("Composer") as "Number of Composers"
from "Track" as t
    left outer join "Album" as a
        on t."AlbumId" = a."AlbumId"
group by a."AlbumId", a."Title"
order by a."AlbumId";
```

# Additional Join Topics

- Self joins: join a table with itself
- Non equi joins: using a non-equality condition

e.g. all different ways to pair up customers

self join

# Self Join Example

- Get all the different ways to pair up customers

```
select
  c1."FirstName" || ' ' || c1."LastName" as "customer_1",
  c2."FirstName" || ' ' || c2."LastName" as "customer_2"
from "Customer" as c1
  cross join "Customer" as c2;
```

almost there (customers paired with themselves for example)

# Self Join Example (continued)

- We do not want customers paired with themselves (non equi join)

```
select
  c1."FirstName" || ' ' || c1."LastName" as "customer_1",
  c2."FirstName" || ' ' || c2."LastName" as "customer_2"
from "Customer" as c1
  inner join "Customer" as c2
    on c1."CustomerId" <> c2."CustomerId";
```

^ can use other comparisons in the "on"

this is a non-equi-join

check for duplicates (in the commutative sense)



# Self Join Example (continued)

- We also do not want duplicates in the commutative sense

```
select
  c1."FirstName" || ' ' || c1."LastName" as "customer_1",
  c2."FirstName" || ' ' || c2."LastName" as "customer_2"
from "Customer" as c1
  inner join "Customer" as c2
    on c1."CustomerId" < c2."CustomerId";
```