Aggregate Function

- An aggregate function takes the value of a field (or an expression over multiple fields) for a set of rows and outputs a single value

- When used alone, an aggregate function reduces a set of rows to a single row
  - In a moment we'll get to *grouping* by field(s)

- Common aggregate functions include **`MAX`**, **`MIN`**, **`SUM`**, **`AVG`**, **`COUNT`**
  - Relational Algebra: $_{\text{<grouping list>}}\mathcal{F}_{\text{<function list>}}(R)$

Continuing Our Example

STUDENT

Name	SSN	Phone	Dorm	Age	GPA
Ben Bayer	305-61-2435	555-1234	1	19	3.21
Chung-cha Kim	422-11-2320	555-9876	2	25	3.53
Barbara Benson	533-69-1238	555-6758	1	19	3.25

GPA
3.21
3.25

Goal: find the GPA of students in MATH650

Approach: cross all rows in STUDENT with all rows in CLASS and keep the GPA of those where STUDENT(SSN)=CLASS(SSN) and CLASS(Class)=MATH650

```
SELECT STUDENT.GPA
FROM STUDENT INNER JOIN CLASS
ON STUDENT.SSN=CLASS.SSN
WHERE CLASS.Class='MATH650';
```

CLASS

SSN	Class
305-61-2435	COMP355
422-11-2320	COMP355
533-69-1238	MATH650
305-61-2435	MATH650
422-11-2320	BIOL110

Now Take the Average!

STUDENT

Name	SSN	Phone	Dorm	Age	GPA
Ben Bayer	305-61-2435	555-1234	1	19	3.21
Chung-cha Kim	422-11-2320	555-9876	2	25	3.53
Barbara Benson	533-69-1238	555-6758	1	19	3.25

aGPA
3.23

Goal: find the average GPA of students in MATH650

Approach: cross all rows in STUDENT with all rows in CLASS and keep the GPA of those where STUDENT(SSN)=CLASS(SSN) and CLASS(Class)=MATH650, average result set

```
SELECT AVG(STUDENT.GPA) AS aGPA
FROM STUDENT INNER JOIN CLASS
ON STUDENT.SSN=CLASS.SSN
WHERE CLASS.Class='MATH650';
```

CLASS

SSN	Class
305-61-2435	COMP355
422-11-2320	COMP355
533-69-1238	MATH650
305-61-2435	MATH650
422-11-2320	BIOL110

Now Take the Average!

STUDENT

Name	SSN	Phone	Dorm	Age	GPA
Ben Bayer	305-61-2435	555-1234	1	19	3.21
Chung-cha Kim	422-11-2320	555-9876	2	25	3.53
Barbara Benson	533-69-1238	555-6758	1	19	3.25

aGPA
3.23

Goal: find the average GPA of students in MATH650

Approach: cross all rows in STUDENT with all rows in CLASS and keep the GPA of those where STUDENT(SSN)=CLASS(SSN) and CLASS(Class)=MATH650, average result set

$$J \leftarrow STUDENT \bowtie_{STUDENT.SSN=CLASS.SSN} CLASS$$
$$S \leftarrow \sigma_{CLASS.Class='MATH650'}(J)$$
$$A \leftarrow \mathscr{F}_{AVG\ Student.GPA}(S)$$
$$RES \leftarrow \rho_{(aGPA)}(A)$$

CLASS

SSN	Class
305-61-2435	COMP355
422-11-2320	COMP355
533-69-1238	MATH650
305-61-2435	MATH650
422-11-2320	BIOL110

SQL: Examples

- Get the number of tracks for an album

```
SELECT COUNT(*) AS num_tracks FROM track WHERE AlbumId=1;
```

  - **`COUNT(*)`** = number of rows
  - **`COUNT(field)`** = number of non-NULL values
  - **`COUNT(DISTINCT field)`** = number of distinct values of a field

- Compute the total cost of an album

```
SELECT SUM(UnitPrice) AS total_cost FROM track WHERE AlbumId=1;
```

- Get the min/max/average track unit price overall

```
SELECT MIN(UnitPrice) AS min_price FROM track;
SELECT MAX(UnitPrice) AS max_price FROM track;
SELECT AVG(UnitPrice) AS avg_price FROM track;

SELECT MIN(UnitPrice) AS min_price, MAX(UnitPrice) AS max_price,
AVG(UnitPrice) AS avg_price FROM track;
```

SQL: Grouping

The **GROUP BY** statement allows you to define subgroups for aggregate functions. The **GROUP BY** attribute list should be a subset of **SELECT** list.

```
SELECT [DISTINCT] <attribute list>
FROM <table list>
[WHERE <condition list>]
[GROUP BY <attribute list>]
[ORDER BY <attribute-order list>];
```

Example: track price stats by media type

```
SELECT mt.Name AS media_type, MIN(t.UnitPrice) AS min_price,
       MAX(t.UnitPrice) AS max_price, AVG(t.UnitPrice) AS avg_price
FROM track t INNER JOIN MediaType mt ON t.MediaTypeId=mt.MediaTypeId
GROUP BY mt.Name
ORDER BY avg_price DESC, mt.Name ASC;
```

Conceptually

```
SELECT mt.Name AS media_type, MIN(t.UnitPrice) AS min_price,
       MAX(t.UnitPrice) AS max_price, AVG(t.UnitPrice) AS avg_price
FROM track t INNER JOIN MediaType mt ON t.MediaTypeId=mt.MediaTypeId
GROUP BY mt.Name
ORDER BY avg_price DESC, mt.Name ASC;
```

```
SELECT *
FROM track t INNER JOIN MediaType mt ON t.MediaTypeId=mt.MediaTypeId
ORDER BY mt.Name ASC;
```

	TrackId	Name	AlbumId	MediaTypeId	GenreId	Composer	Milliseconds	Bytes	UnitPrice	MediaTypeId	Name
1	1	For Those About To Rock (We Salute You)	1	1	1	Angus Young, Malcolm Young, Brian Johnson	343719	11170334	0.99	1	MPEG audio file
2	6	Put The Finger On You	1	1	1	Angus Young, Malcolm Young, Brian Johnson	205662	6713451	0.99	1	MPEG audio file
3	7	Let's Get It Up	1	1	1	Angus Young, Malcolm Young, Brian Johnson	233926	7636561	0.99	1	MPEG audio file
4	2	Balls to the Wall	2	2	1		342562	5510424	0.99	2	Protected AAC audio file
5	3	Fast As a Shark	3	2	1	F. Baltes, S. Kaufman, U. Dirkscneider & W. Hoffman	230619	3990994	0.99	2	Protected AAC audio file
6	4	Restless and Wild	3	2	1	F. Baltes, R.A. Smith-Diesel, S. Kaufman, U. Dirkscneider & W. Hoffman	252051	4331779	0.99	2	Protected AAC audio file
7	5	Princess of the Dawn	3	2	1	Deaffy & R.A. Smith-Diesel	375418	6290521	0.99	2	Protected AAC audio file

...

Relational Algebra

```sql
SELECT mt.Name AS media_type, MIN(t.UnitPrice) AS min_price,
       MAX(t.UnitPrice) AS max_price, AVG(t.UnitPrice) AS avg_price
FROM track t INNER JOIN MediaType mt ON t.MediaTypeId=mt.MediaTypeId
GROUP BY mt.Name
ORDER BY avg_price DESC, mt.Name ASC;
```

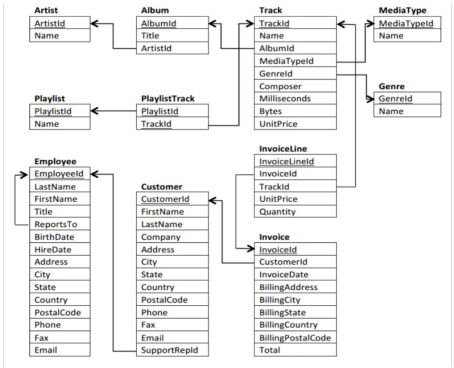
$$J \leftarrow \rho_{t}(Track) \bowtie_{t.MediaTypeId=mt.MediaTypeId} \rho_{mt}(MediaType)$$

$$A \leftarrow_{mt.Name} \mathscr{F}_{mt.Name,\ \text{MIN}\ t.UnitPrice,\ \text{MAX}\ t.UnitPrice,\ \text{AVG}\ t.UnitPrice}(J)$$

$$R \leftarrow \rho_{(media\_type, min\_price, max\_price, avg\_price)}(A)$$

$$RES \leftarrow \tau_{avg\_price\ \text{DESC},\ mt.Name}(R)$$

Grouped Aggregation (1)



	BillingCity	BillingState	avg_total	sum_total	ct
1	Fort Worth	TX	6.80285714285714	47.62	7
2	Chicago	IL	6.23142857142857	43.62	7
3	Salt Lake City	UT	6.23142857142857	43.62	7
4	Madison	WI	6.08857142857143	42.62	7
5	Orlando	FL	5.66	39.62	7
6	Redmond	WA	5.66	39.62	7
7	Cupertino	CA	5.51714285714286	38.62	7
8	Mountain View	CA	5.51714285714286	77.24	14
9	Tucson	AZ	5.37428571428571	37.62	7
10	Boston	MA	5.37428571428571	37.62	7
11	Reno	NV	5.37428571428571	37.62	7
12	New York	NY	5.37428571428571	37.62	7

Get the average, sum, and number of all US invoices, grouped by city and state. Order by average cost (greatest first), then state (alphabetically), then city (alphabetically).

```
SELECT BillingCity, BillingState,
       AVG(Total) AS avg_total, SUM(Total) AS sum_total, COUNT(*) AS ct
FROM invoice
WHERE BillingCountry='USA'
GROUP BY BillingCity, BillingState
ORDER BY avg_total DESC, BillingState ASC, BillingCity ASC;
```

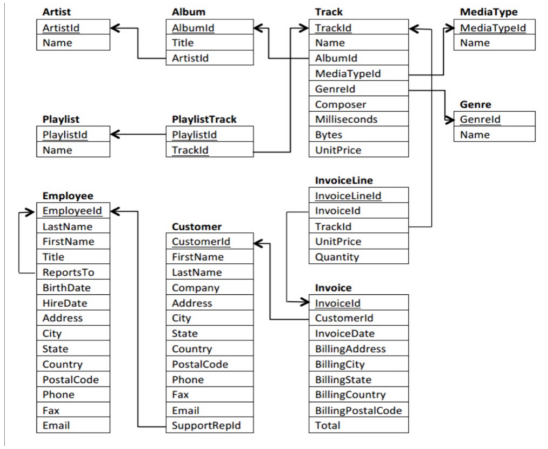
$$S \leftarrow \sigma_{BillingCountry='USA'}(invoice)$$

$$A \leftarrow {}_{BillingCity,BillingState}\mathcal{F}_{BillingCity,BillingState,AVG\ Total,SUM\ Total,COUNT(*)}(S)$$

$$R \leftarrow \rho_{(BillingCity,BillingState,avg\_total,sum\_total,ct)}(A)$$

$$RES \leftarrow \tau_{avg\_total\ DESC,BillingState,BillingCity}(R)$$

Grouped Aggregation (2)



	InvoiceId	total
1	404	25.86
2	299	23.86
3	96	21.86
4	194	21.86
5	89	18.86
6	201	18.86
7	88	17.91
8	306	16.86
9	313	16.86
10	103	15.86
11	208	15.86
12	193	14.91

Using only the invoiceline table, compute the total cost of each order, sorted by total (greatest first), then invoice id (smallest first).

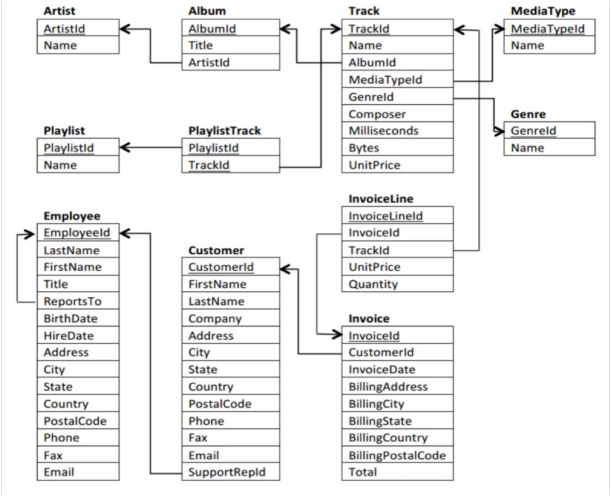
```
SELECT InvoiceId, SUM(UnitPrice*Quantity) AS total
FROM invoiceline
GROUP BY InvoiceId
ORDER BY total DESC, InvoiceId ASC;
```

$$A \leftarrow {}_{InvoiceId}\mathcal{F}_{InvoiceId,SUM\ (UnitPrice*Quantity)}(invoiceline)$$

$$R \leftarrow \rho_{(InvoiceId,total)}(A)$$

$$RES \leftarrow \tau_{total\ DESC,InvoiceId}(R)$$

Grouped Aggregation (3)

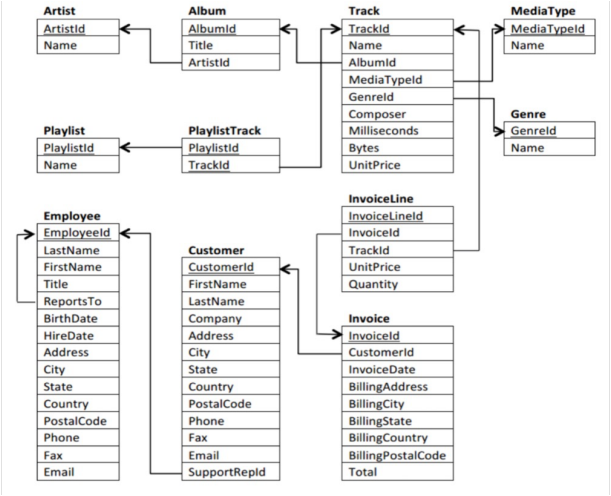


	TrackId	Name	Title	num_sold
1	430	I'm Going Slightly Mad	Greatest Hits II	2
2	2263	Somebody To Love	Greatest Hits I	2
3	2272	We Are The Champions	News Of The World	2
4	2259	You're My Best Friend	Greatest Hits I	2
5	419	A Kind Of Magic	Greatest Hits II	1
6	2274	All Dead, All Dead	News Of The World	1
7	2255	Another One Bites The Dust	Greatest Hits I	1
8	2258	Bicycle Race	Greatest Hits I	1
9	2254	Bohemian Rhapsody	Greatest Hits I	1
10	426	Breakthru	Greatest Hits II	1
11	2257	Fat Bottomed Girls	Greatest Hits I	1
12	2276	Fight From The Inside	News Of The World	1

Generate a ranked list of Queen's best selling tracks. Display the track id, track name, and album name, along with number of tracks sold, sorted by tracks sold (greatest first), then by track name (alphabetical).

```
SELECT invoiceline.TrackId, track.Name, album.Title,
       SUM(invoiceline.Quantity) AS num_sold
FROM ((invoiceline INNER JOIN track ON invoiceline.TrackId=track.TrackId)
INNER JOIN album ON track.AlbumId=album.AlbumId)
INNER JOIN artist ON album.ArtistId=artist.ArtistId
WHERE artist.Name='Queen'
GROUP BY invoiceline.TrackId
ORDER BY num_sold DESC, track.Name ASC;
```

Grouped Aggregation (3-RA)



	TrackId	Name	Title	num_sold
1	430	I'm Going Slightly Mad	Greatest Hits II	2
2	2263	Somebody To Love	Greatest Hits I	2
3	2272	We Are The Champions	News Of The World	2
4	2259	You're My Best Friend	Greatest Hits I	2
5	419	A Kind Of Magic	Greatest Hits II	1
6	2274	All Dead, All Dead	News Of The World	1
7	2255	Another One Bites The Dust	Greatest Hits I	1
8	2258	Bicycle Race	Greatest Hits I	1
9	2254	Bohemian Rhapsody	Greatest Hits I	1
10	426	Breakthru	Greatest Hits II	1
11	2257	Fat Bottomed Girls	Greatest Hits I	1
12	2276	Fight From The Inside	News Of The World	1

Generate a ranked list of Queen's best selling tracks. Display the track id, track name, and album name, along with number of tracks sold, sorted by tracks sold (greatest first), then by track name (alphabetical).

$$J1 \leftarrow invoiceline \bowtie_{invoiceline.TrackId=track.TrackId} track$$
$$J2 \leftarrow J1 \bowtie_{track.AlbumId=album.AlbumId} album$$
$$J3 \leftarrow J2 \bowtie_{album.ArtistId=artist.ArtistId} artist$$
$$S \leftarrow \sigma_{artist.Name='Queen'}(J3)$$
$$A \leftarrow_{invoiceline.TrackId} \mathscr{F}_{invoiceline.TrackId,track.Name,album.Title,SUM\ invoiceline.Quantity}(S)$$
$$R \leftarrow \rho_{(TrackId,Name,Title,num\_sold)}(A)$$
$$RES \leftarrow \tau_{num\_sold\ DESC,Name}(R)$$

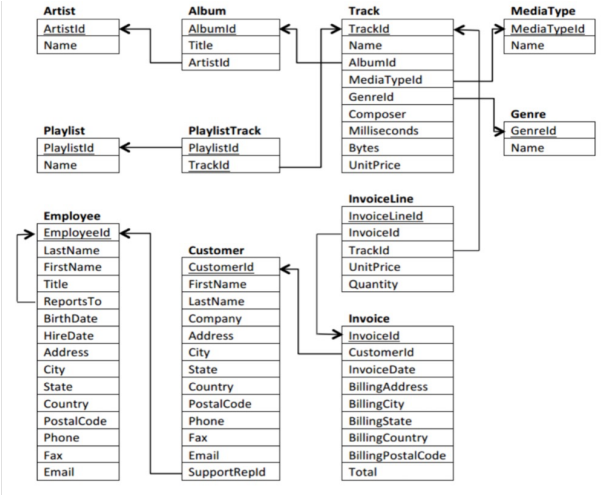
SQL: HAVING

The **HAVING** statement allows you to place constraint(s), similar to **WHERE**, that use aggregate functions (separate by **AND**/**OR**)

- Same as SELECT condition in relational algebra, but has efficiency implications in DBMS

```
SELECT [DISTINCT] <attribute list>
FROM <table list>
[WHERE <condition list>]
[GROUP BY <attribute list>]
[HAVING <condition list>]
[ORDER BY <attribute-order list>];
```

Aggregation (4)



	TrackId	Name	Title	num_sold
1	430	I'm Going Slightly Mad	Greatest Hits II	2
2	2263	Somebody To Love	Greatest Hits I	2
3	2272	We Are The Champions	News Of The World	2
4	2259	You're My Best Friend	Greatest Hits I	2

Generate a ranked list of Queen's best selling tracks. Display the track id, track name, and album name, along with number of tracks sold, sorted by tracks sold (greatest first), then by track name (alphabetical). Only show those tracks that have sold at least twice.

```
SELECT invoiceline.TrackId, track.Name, album.Title,
       SUM(invoiceline.Quantity) AS num_sold
FROM ((invoiceline INNER JOIN track ON invoiceline.TrackId=track.TrackId)
INNER JOIN album ON track.AlbumId=album.AlbumId)
INNER JOIN artist ON album.ArtistId=artist.ArtistId
WHERE artist.Name='Queen'
GROUP BY invoiceline.TrackId
HAVING SUM(invoiceline.Quantity)>=2
ORDER BY num_sold DESC, track.Name ASC;
```

Query in a Query

A feature of SQL is its *composability* – the result(s) of one query, which is a set of rows/columns, can be used by another

- Termed inner/nested query or subquery

Most common locations

- **`SELECT`** (returns a value for an attribute)
- **`FROM`** (becomes a "table" to query/join)
- **`WHERE`** (serves as part of a constraint)

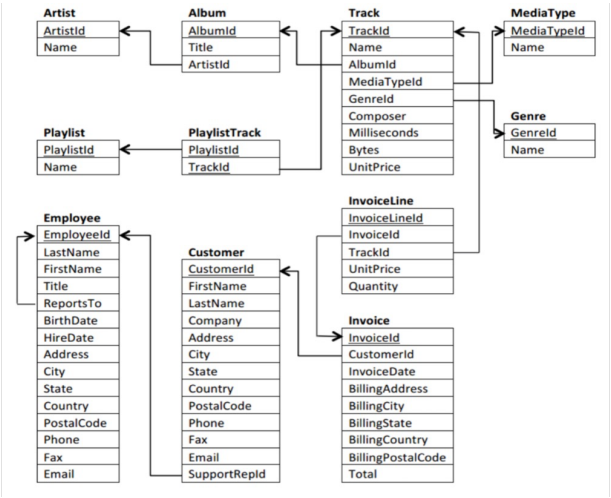
Notes about Subqueries

- Tip: when designing subqueries, work inside out – come up with each query separately, then piece them together
  - Helps with debugging

- A **correlated** subquery is an *inner* query that references a value from an *outer* query
  - The inner query will be run once for *every* tuple of the outer query (i.e. slow!)
  - Common when using as `SELECT` clause

- Don't use `ORDER BY` in inner queries (some DBMSs don't allow, typically wasteful anyhow)

Example: `WHERE`



	TrackId	Name	AlbumId	MediaTypeId	GenreId	Composer	Milliseconds	Bytes	UnitPrice
1	38	All I Really Want	6	1	1	Alanis Morissette & Glenn Ballard	284891	9375567	0.99
2	39	You Oughta Know	6	1	1	Alanis Morissette & Glenn Ballard	249234	8196916	0.99
3	40	Perfect	6	1	1	Alanis Morissette & Glenn Ballard	188133	6145404	0.99
4	41	Hand In My Pocket	6	1	1	Alanis Morissette & Glenn Ballard	221570	7224246	0.99
5	42	Right Through You	6	1	1	Alanis Morissette & Glenn Ballard	176117	5793082	0.99
6	43	Forgiven	6	1	1	Alanis Morissette & Glenn Ballard	300355	9753256	0.99
7	44	You Learn	6	1	1	Alanis Morissette & Glenn Ballard	239699	7824837	0.99
8	45	Head Over Feet	6	1	1	Alanis Morissette & Glenn Ballard	267493	8758008	0.99
9	46	Mary Jane	6	1	1	Alanis Morissette & Glenn Ballard	280607	9163588	0.99
10	47	Ironic	6	1	1	Alanis Morissette & Glenn Ballard	229825	7598866	0.99
11	48	Not The Doctor	6	1	1	Alanis Morissette & Glenn Ballard	227631	7604601	0.99
12	49	Wake Up	6	1	1	Alanis Morissette & Glenn Ballard	293485	9703359	0.99
13	50	You Oughta Know (Alternate)	6	1	1	Alanis Morissette & Glenn Ballard	491885	16008629	0.99

Get all track information for the album Jagged Little Pill (do not use a join)

```
SELECT t.*
FROM track t
WHERE t.AlbumId = (
   SELECT a.AlbumId
   FROM album a
   WHERE a.Title='Jagged Little Pill'
);
```

Notes

1. The subquery needs to return a *single* value for the = to make sense
2. Not correlated!

How the Query Works Conceptually

```
SELECT t.*
FROM track t
WHERE t.AlbumId = (
   SELECT a.AlbumId
   FROM album a
   WHERE a.Title='Jagged Little Pill'
);
```

Inner Query

AlbumId
6

```
SELECT t.*
FROM track t
WHERE t.AlbumId = 6;
```

$$INNER \leftarrow \pi_{AlbumId}(\sigma_{a.Title='Jagged\ Little\ Pill'}(\rho_a(album)))$$

$$OUTER \leftarrow \sigma_{t.AlbumId=INNER}(\rho_t(track))$$

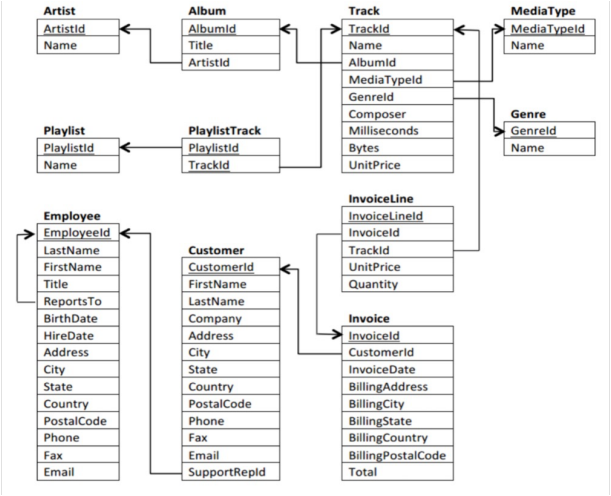
Notes about Subqueries and `WHERE`

For most operators, the subquery will need to return a single value

Other operators:

- [`NOT`] `IN` = query returns a single column of options
- [`NOT`] `EXISTS` = checks if query returns at least a single row
- <op> `ALL` = true if <op> returns true for *all* results (single field)
- <op> `ANY`/`SOME` = true if <op> returns true for *any* result (single field)

Nesting Example: WHERE



	TrackId	Name	AlbumId	MediaTypeId	GenreId	Composer	Milliseconds	Bytes	UnitPrice
1	419	A Kind Of Magic	36	1	1	Roger Taylor	262608	8689618	0.99
2	420	Under Pressure	36	1	1	Queen & David Bowie	236617	7739042	0.99
3	421	Radio GA GA	36	1	1	Roger Taylor	343745	11358573	0.99
4	422	I Want It All	36	1	1	Queen	241684	7876564	0.99
5	423	I Want To Break Free	36	1	1	John Deacon	259108	8552861	0.99
6	424	Innuendo	36	1	1	Queen	387761	12664591	0.99
7	425	It's A Hard Life	36	1	1	Freddie Mercury	249417	8112242	0.99
8	426	Breakthru	36	1	1	Queen	249234	8150479	0.99
9	427	Who Wants To Live Forever	36	1	1	Brian May	297691	9577577	0.99
10	428	Headlong	36	1	1	Queen	273057	8921404	0.99
11	429	The Miracle	36	1	1	Queen	294974	9671923	0.99
12	430	I'm Going Slightly Mad	36	1	1	Queen	248032	8192339	0.99
13	431	The Invisible Man	36	1	1	Queen	238994	7920353	0.99

Get all track information for the artist Queen (do not use a join)

```
SELECT t.*
FROM track t
WHERE t.AlbumId IN (
   SELECT alb.AlbumId
   FROM album alb
   WHERE alb.ArtistId = (
        SELECT art.ArtistId
        FROM artist art
        WHERE art.Name='Queen'
   )
);
```

Notes

1. **Not correlated!**

How the Query Works Conceptually

```
SELECT t.*
FROM track t
WHERE t.AlbumId IN (
   SELECT alb.AlbumId
   FROM album alb
   WHERE alb.ArtistId = (
        SELECT art.ArtistId
        FROM artist art
        WHERE art.Name='Queen'
   )
);
```

	ArtistId
1	51

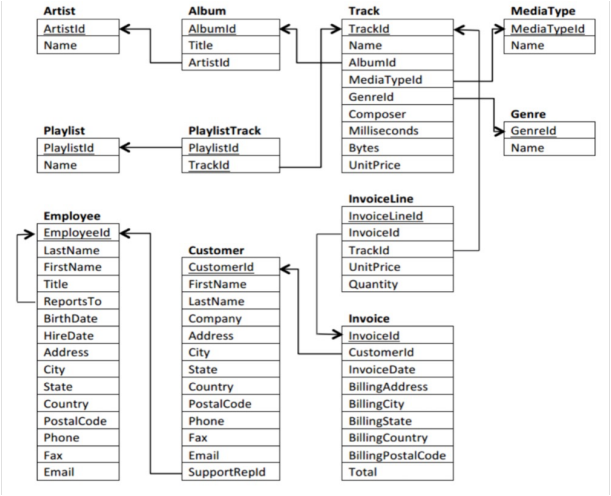
```
SELECT t.*
FROM track t
WHERE t.AlbumId IN (
   SELECT alb.AlbumId
   FROM album alb
   WHERE alb.ArtistId = 51
);
```

	AlbumId
1	36
2	185
3	186

```
SELECT t.*
FROM track t
WHERE t.AlbumId IN (36, 185, 186);
```

$$IN2 \leftarrow \pi_{art.ArtistId}(\sigma_{art.Name='Queen'}(\rho_{art}(artist)))$$
$$IN1 \leftarrow \pi_{alb.AlbumId}(\sigma_{alb.ArtistId=IN2}(\rho_{alb}(album)))$$
$$OUT \leftarrow \sigma_{t.AlbumId\ IN\ IN2}(\rho_{t}(track))$$

Example: SELECT



	artist_name	album_ct
1	Santana	3
2	Santana Feat. Dave Matthews	0
3	Santana Feat. Eagle-Eye Cherry	0
4	Santana Feat. Eric Clapton	0
5	Santana Feat. Everlast	0
6	Santana Feat. Lauryn Hill & Cee-Lo	0
7	Santana Feat. Maná	0
8	Santana Feat. Rob Thomas	0
9	Santana Feat. The Project G&B	0

For each artist starting with "Santana", get the number of albums, sorted by count (greatest first), then artist (alphabetical)

```
SELECT art.Name AS artist_name,
       (
            SELECT COUNT(*)
            FROM album alb
            WHERE alb.ArtistId=art.ArtistId
        ) AS album_ct
FROM artist art
WHERE art.Name LIKE 'Santana%'
ORDER BY album_ct DESC, art.Name;
```

Notes

1. The subquery needs to return a *single* value for each tuple generated
2. Correlated subquery!! :(

How the Query Works Conceptually

```
SELECT art.Name AS artist_name,
       (
            SELECT COUNT(*)
            FROM album alb
            WHERE alb.ArtistId=art.ArtistId
        ) AS album_ct
FROM artist art
WHERE art.Name LIKE 'Santana%'
ORDER BY album_ct DESC, art.Name;
```

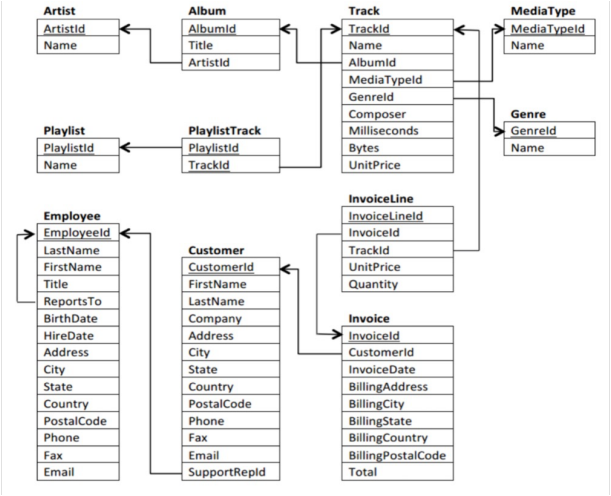
Correlated - one query per row to fill in album_ct column!

```
SELECT COUNT(*)
FROM album alb
WHERE alb.ArtistId=59;
                  =60;
                  …
```

```
SELECT *
FROM artist art
WHERE art.Name LIKE 'Santana%';
```

	ArtistId	Name
1	59	Santana
2	60	Santana Feat. Dave Matthews
3	61	Santana Feat. Everlast
4	62	Santana Feat. Rob Thomas
5	63	Santana Feat. Lauryn Hill & Cee-Lo
6	64	Santana Feat. The Project G&B
7	65	Santana Feat. Maná
8	66	Santana Feat. Eagle-Eye Cherry
9	67	Santana Feat. Eric Clapton

[Better] Example: FROM



	artist_name	album_ct
1	Santana	3
2	Santana Feat. Dave Matthews	0
3	Santana Feat. Eagle-Eye Cherry	0
4	Santana Feat. Eric Clapton	0
5	Santana Feat. Everlast	0
6	Santana Feat. Lauryn Hill & Cee-Lo	0
7	Santana Feat. Maná	0
8	Santana Feat. Rob Thomas	0
9	Santana Feat. The Project G&B	0

For each artist starting with Santana, get the number of albums, sorted by count (greatest first), then artist (alphabetical)

```
SELECT artist_name, COUNT(q1.AlbumId) AS album_ct
FROM
(
   SELECT art.ArtistId AS artist_id, art.Name AS artist_name, alb.AlbumId
   FROM artist art LEFT JOIN album alb ON art.ArtistId=alb.ArtistId
   WHERE art.Name LIKE 'Santana%'
) q1
GROUP BY artist_id
ORDER BY album_ct DESC, artist_name;
```

How the Query Works Conceptually

```
SELECT artist_name, COUNT(q1.AlbumId) AS album_ct
FROM
(
   SELECT art.ArtistId AS artist_id, art.Name AS artist_name, alb.AlbumId
   FROM artist art LEFT JOIN album alb ON art.ArtistId=alb.ArtistId
   WHERE art.Name LIKE 'Santana%'
) q1
GROUP BY artist_id
ORDER BY album_ct DESC, artist_name;
```

q1

	artist_id	artist_name	AlbumId
1	59	Santana	46
2	59	Santana	197
3	59	Santana	198
4	60	Santana Feat. Dave Matthews	*NULL*
5	61	Santana Feat. Everlast	*NULL*
6	62	Santana Feat. Rob Thomas	*NULL*
7	63	Santana Feat. Lauryn Hill & Cee-Lo	*NULL*
8	64	Santana Feat. The Project G&B	*NULL*
9	65	Santana Feat. Maná	*NULL*
10	66	Santana Feat. Eagle-Eye Cherry	*NULL*
11	67	Santana Feat. Eric Clapton	*NULL*

```
SELECT artist_name, COUNT(q1.AlbumId) AS album_ct
FROM q1
GROUP BY artist_id
ORDER BY album_ct DESC, artist_name;
```

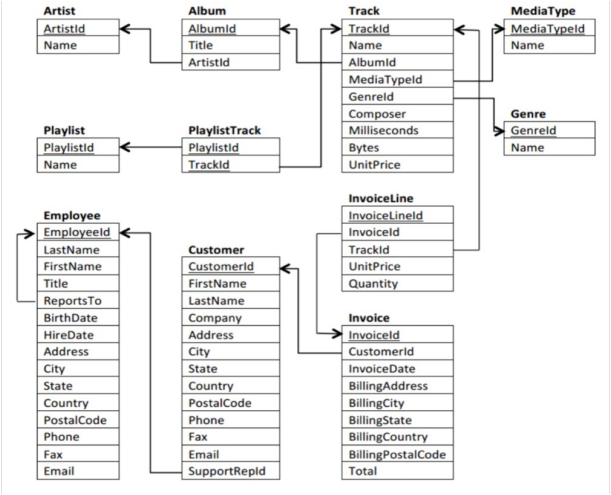
	artist_name	album_ct
1	Santana	3
2	Santana Feat. Dave Matthews	0
3	Santana Feat. Eagle-Eye Cherry	0
4	Santana Feat. Eric Clapton	0
5	Santana Feat. Everlast	0
6	Santana Feat. Lauryn Hill & Cee-Lo	0
7	Santana Feat. Maná	0
8	Santana Feat. Rob Thomas	0
9	Santana Feat. The Project G&B	0

Notes about Subqueries and `FROM`

- When using one or more subqueries in the `FROM` clause, remember two important items
  - The subquery must be enclosed within parentheses
  - The subquery must have a name (e.g. `q1` in the previous example), which is indicated just after the close parenthesis

- The name can be used to refer to columns in the subquery via the dot notation (e.g. subqueryname.columnname) – this is required if the column name is not unique

Nesting Example: FROM



	min_q	max_q	avg_q	num_customers
1	36	38	37.9661016949153	59

Find the minimum, maximum, and average number of tracks ordered per customer (across all invoices). Also include the total number of customers.

```
SELECT MIN(q2.sum_q) AS min_q, MAX(q2.sum_q) AS max_q, AVG(q2.sum_q) AS avg_q,
       COUNT(*) AS num_customers
FROM
   (SELECT q1.CustomerId, SUM(Quantity) AS sum_q
    FROM
        (SELECT i.CustomerId, il.Quantity
         FROM invoice i NATURAL JOIN invoiceline il
        ) q1
    GROUP BY q1.CustomerId
   ) q2;
```

How the Query Works Conceptually

```
SELECT MIN(q2.sum_q) AS min_q, MAX(q2.sum_q) AS max_q, AVG(q2.sum_q) AS avg_q,
       COUNT(*) AS num_customers
FROM
   (SELECT q1.CustomerId, SUM(Quantity) AS sum_q
    FROM
        (SELECT i.CustomerId, il.Quantity
         FROM invoice i NATURAL JOIN invoiceline il
        ) q1
    GROUP BY q1.CustomerId
   ) q2;
```

q2

	CustomerId	sum_q
1	1	38
2	2	38
3	3	38
4	4	38
5	5	38
6	6	38
7	7	38
8	8	38

...

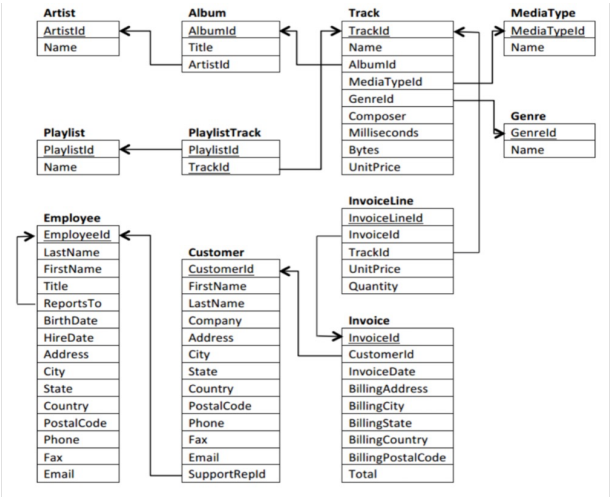
q1

	CustomerId	Quantity
1	2	1
2	2	1
3	4	1
4	4	1
5	4	1
6	4	1
7	8	1
8	8	1

...

	min_q	max_q	avg_q	num_customers
1	36	38	37.9661016949153	59

Subquery (1)

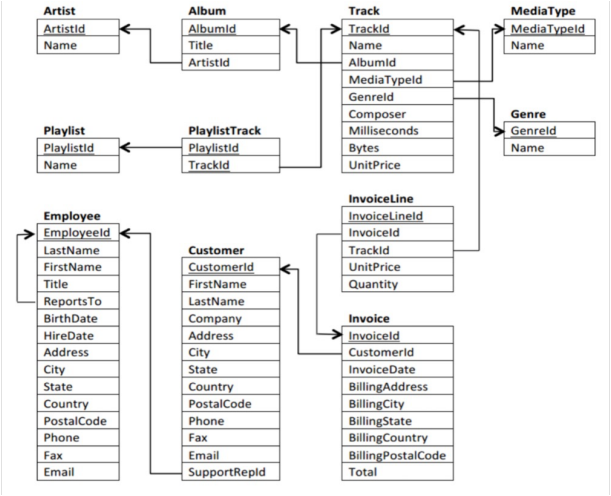


	FirstName	LastName	total_spent
1	Helena	Holý	49.62
2	Richard	Cunningham	47.62
3	Luis	Rojas	46.62
4	Ladislav	Kovács	45.62
5	Hugh	O'Reilly	45.62
6	Julia	Barnett	43.62
7	Frank	Ralston	43.62
8	Fynn	Zimmermann	43.62
9	Astrid	Gruber	42.62
10	Victor	Stevens	42.62
11	Terhi	Hämäläinen	41.62
12	Isabelle	Mercier	40.62
13	Johannes	Van der Berg	40.62
14	František	Wichterlová	40.62

Find the highest spending customers: get a ranked list of customers (first name, last name) who have spent at least $40, sorted by amount spent (greatest first), then last name, then first name

```
SELECT * FROM (
   SELECT c.FirstName, c.LastName, (
        SELECT SUM(i.Total)
        FROM invoice i
        WHERE c.CustomerId=i.CustomerId
   ) AS total_spent
   FROM customer c) q1
WHERE q1.total_spent>=40
ORDER BY q1.total_spent DESC, q1.LastName ASC, q1.FirstName ASC;
```

Subquery (2)



	g_name	g_ct	g_percentage
1	Rock	1297	37.0254067941764
2	Latin	579	16.5286896945475
3	Metal	374	10.6765629460462
4	Alternative & Punk	332	9.4775906365972
5	Jazz	130	3.71110476734228
6	TV Shows	93	2.65486725663717
7	Blues	81	2.31230373965173
8	Classical	74	2.11247502141022
9	Drama	64	1.82700542392235
10	R&B/Soul	61	1.74136454467599
11	Reggae	58	1.65572366542963
12	Pop	48	1.37025406794176
13	Soundtrack	43	1.22751926919783

Create a report of the distribution of tracks into genres. The result set should list each genre by name, the number of tracks of that genre, and the percentage of overall tracks for that genre. The rows should be sorted by the percentage (greatest first), then genre name (alphabetically).

```
SELECT x.Name AS g_name, x.g_ct AS g_ct, (100.0 * g_ct / ct) AS g_percentage
FROM (SELECT *, (SELECT COUNT(*) FROM track t1 WHERE t1.GenreId=g.GenreId) AS g_ct,
                (SELECT COUNT(*) FROM track t2) AS ct
      FROM genre g) x
ORDER BY g_percentage DESC, g_name ASC;
```

Exercise

What is the purpose of the following query?

```
SELECT
   t.name AS trackName,
   COUNT(*) AS trackSales
FROM
   Track t INNER JOIN InvoiceLine il
      ON t.TrackId=il.TrackId
WHERE
   t.name IN ('Iron Maiden', 'Sanctuary', 'Time')
GROUP BY
   t.Name
```

	trackName	trackSales
1	Iron Maiden	3
2	Sanctuary	3
3	Time	1

Exercise

What is the purpose of the following query?

```
SELECT
   t.name AS trackName,
   COUNT(*) AS trackSales
FROM
   Track t INNER JOIN InvoiceLine il
      ON t.TrackId=il.TrackId
WHERE
   t.name IN ('Iron Maiden', 'Sanctuary', 'Time')
GROUP BY
   t.TrackId
```

	trackName	trackSales
1	Iron Maiden	1
2	Sanctuary	1
3	Iron Maiden	1
4	Sanctuary	1
5	Sanctuary	1
6	Iron Maiden	1
7	Time	1

Challenge

You are **not** allowed to include columns in SELECT that are not (a) part of GROUP BY or (b) part of an aggregate expression

- Some DBMSs follow this policy

How do you re-write the following query?

Exercise

```
SELECT
   t.name AS trackName,
   COUNT(*) AS trackSales
FROM
   Track t INNER JOIN InvoiceLine il
      ON t.TrackId=il.TrackId
WHERE
   t.name IN ('Iron Maiden', 'Sanctuary', 'Time')
GROUP BY
   t.TrackId
```

	trackName	trackSales
1	Iron Maiden	1
2	Sanctuary	1
3	Iron Maiden	1
4	Sanctuary	1
5	Sanctuary	1
6	Iron Maiden	1
7	Time	1

(An) Answer

```
SELECT
    t.Name AS trackName,
    a.trackSales
FROM
    Track t INNER JOIN
        (SELECT
            t.TrackId AS trackId,
            COUNT(*) AS trackSales
        FROM
            Track t INNER JOIN InvoiceLine il
                ON t.TrackId=il.TrackId
        WHERE
            t.name IN ('Iron Maiden', 'Sanctuary', 'Time')
        GROUP BY
            t.TrackId) a
    ON t.TrackId=a.trackId
```

	trackName	trackSales
1	Iron Maiden	1
2	Sanctuary	1
3	Iron Maiden	1
4	Sanctuary	1
5	Sanctuary	1
6	Iron Maiden	1
7	Time	1

(Another) Answer

```
SELECT
    a.trackName,
    a.trackSales
FROM
    (SELECT
        t.Name AS trackName,
        t.AlbumId AS trackAlbum,
        COUNT(*) AS trackSales
    FROM
        Track t INNER JOIN InvoiceLine il
            ON t.TrackId=il.TrackId
    WHERE
        t.name IN ('Iron Maiden', 'Sanctuary', 'Time')
    GROUP BY
        t.Name, t.AlbumId) a
```

	trackName	trackSales
1	Iron Maiden	1
2	Iron Maiden	1
3	Iron Maiden	1
4	Sanctuary	1
5	Sanctuary	1
6	Sanctuary	1
7	Time	1

Inserting Rows

- Insert all attributes, in same order as table

```
INSERT INTO table_name
VALUES (a, b, … n);
```

- Insert a subset of attributes (not assigned = **NULL**)

```
INSERT INTO table_name (a1, a2, … an)
VALUES (a, b, … n)[, (a2, b2, … n2), …];
```

- Insert via query

```
INSERT INTO table_name (a1, a2, … an)
SELECT a1, a2, … an FROM …
```

Updating Rows

General syntax

```
UPDATE table_name
SET <attribute=value list>
[WHERE <condition list>];
```

- Attribute=value is comma-separated
- Condition list may result in more than one rows being updated via a single statement

Deleting Rows

General syntax

```
DELETE FROM table_name
[WHERE <condition list>];
```

- Condition list may result in more than one rows being deleted via a single statement
- No condition = clear table (*truncate*)

Summary

- You have now learned most of the DML components of SQL
  - **`SELECT`**: get stuff out
  - **`INSERT`**: add row(s)
  - **`UPDATE`**: change existing row(s)
  - **`DELETE`**: remove row(s)

- While using **`SELECT`** you learned about attribute ordering/renaming (**`AS`**), row filtering (**`WHERE`**) and sorting (**`ORDER BY`**), table joining (**`FROM`** + **`JOIN`**/**`ON`**), grouped aggregation (**`GROUP BY`** + **`FN`** + **`HAVING`**), set operations on multiple queries (e.g. **`UNION`**), and subqueries (**`SELECT`** within **`SELECT`**)

- You have also learned the basic relational algebra operators associated with SELECT (σ,π,ρ,τ,δ,$\bowtie$,$\mathcal{F}$)