Database Conditionals, Grouping, and Joins

Motivating questions

- How can we narrow the sets of records that get returned from a query?
- How can we make statistical queries across different groupings of records?
- How can we sample or order our output results?
- How can we integrate data from another source with our database?

Narrowing select statements with conditional clauses

WHERE IN

- `WHERE` *`attribute`* `IN` is used to select rows that are satisfied by a set of `WHERE` conditions on the same attribute:

```
SELECT *
FROM Offices
WHERE State IN ('CO', 'UT', 'TX');
```

- Equivalent to:

```
SELECT *
FROM Offices
WHERE State = 'CO' OR State = 'UT'
     OR State = 'TX';
```

OfficeNbr	City	State	Region	Target	Sales	Phone
1	Denver	CO	West	3000000.00	130000.00	970.586.3341
57	Dallas	TX	West	0.00	0.00	214.781.5342

BETWEEN keyword

- Select records where the attribute value is between two numbers using BETWEEN
- Range is inclusive and also works with time and date data

```
SELECT *
FROM SalesReps
WHERE Quota BETWEEN 50000 AND 100000;
```

RepNbr	Name	RepOffice	Quota	Sales
53	Bill Smith	1	100000.00	0.00
89	Jen Jones	2	50000.00	130000.00

```
SELECT *
FROM SalesReps
WHERE Quota NOT BETWEEN 50000 AND 100000;
```

RepNbr	Name	RepOffice	Quota	Sales

NULL values

- Empty or missing values are stored in tables as `NULL`
- `NULL` values evaluate to NOT TRUE in all cases
- Insert a new Sales Representative with a NULL Quota:

```
INSERT INTO SalesReps
VALUES ('25', 'Chris', '1', NULL, 1000.0)
```

RepNbr	Name	RepOffice	Quota	Sales
53	Bill Smith	1	100000.00	0.00
89	Jen Jones	2	50000.00	130000.00
25	Chris	1	NULL	1000.00

NULLs

- The following two sets of queries will not return all sales reps:

```
SELECT Name
FROM SalesReps
WHERE Sales > Quota;

SELECT Name
FROM SalesReps
WHERE Sales <= Quota;
```

```
SELECT Name
FROM SalesReps
WHERE Sales = Quota;

SELECT Name
FROM SalesReps
WHERE Sales <> Quota;
```

RepNbr	Name	RepOffice	Quota	Sales
53	Bill Smith	1	100000.00	0.00
89	Jen Jones	2	50000.00	130000.00
25	Chris	1	NULL	1000.00

NULLs

- Check for NULLS using `IS`:

```
SELECT Name
FROM SalesReps
WHERE Quota IS NULL;
```

- Check for NULLS using `IS NOT`:

```
SELECT Name
FROM SalesReps
WHERE Quota IS NOT NULL;
```

Key points from lesson

- WHERE IN statements are used to identify records in a table with an attribute matching a value from a specified set of values

- BETWEEN keywords are used to identify records that have values for a particular attribute that fall within a specified range

- When a specific attribute may contain NULL or missing values, special care must be taken when using these conditional clauses

Grouping Data and Statistical Functions

GROUP BY with COUNT(*)

- Find the number of sales for each customer:

```
SELECT Cust, COUNT(*)
FROM Orders
GROUP BY Cust;
```

OrderNbr	Cust	Prod	Qty	Amt	Disc
1	211	Bulldozer	7	31000.00	0.20
2	522	Riveter	2	4000.00	0.30
3	522	Crane	1	500000.00	0.40

Cust	COUNT(*)
211	1
522	2

GROUP BY with COUNT(*) and HAVING

- Return the number of parts from each vendor:

```
SELECT Vendor, COUNT(*)
FROM Parts
GROUP BY Vendor;
```

Vendor	COUNT(*)
A	4
B	1
C	2

PartID	Vendor
123	A
234	A
345	B
362	A
2345	C
3464	A
4533	C

- Return the number of parts from each vendor who sells more than two parts:

```
SELECT Vendor, COUNT(*)
FROM Parts
GROUP BY Vendor
HAVING COUNT(*) > 2;
```

Vendor	COUNT(*)
A	4

Aggregate Statistical Functions in SQL

- Statistical functions are available in SQL to perform analytics
- Commonly used functions include:

Name	Description
AVG()	Return the average value of the argument
COUNT()	Return a count of the number of rows returned
COUNT(DISTINCT)	Return the count of a number of different values
MAX()	Return the maximum value
MIN()	Return the minimum value
STD()	Return the population standard deviation
STDDEV_SAMP()	Return the sample standard deviation
SUM()	Return the sum
VAR_POP()	Return the population standard variance
VAR_SAMP()	Return the sample variance
VARIANCE()	Return the population standard variance

http://dev.mysql.com/doc/refman/5.7/en/group-by-functions.html

Advanced Statistical Functions in SQL

- More advanced statistical functions can be created using the basic statistical functions built into SQL
- Calculate the weighted average of student scores from the table of grades:

```
SELECT SUM(student_score*score_weight)/
SUM(score_weight)
FROM grades;
```

- Get the z-score values for student_score from the table grades:

```
SELECT (student_score-AVG(student_score))/
VARIANCE(student_score)
FROM grades;
```

Key points from lesson

- Built-in statistical functions exist in many implementations of SQL
- Statistical functions can operate on a group of records and return a single value for each group of records created by a GROUP BY clause
- To restrict the output of a GROUP BY clause to results which meet specific conditions, use the HAVING keyword, which is analogous to a WHERE clause

Sorting and Sampling Data

ORDER BY

- Order by one column, ascending, `ASC`, or descending, `DESC`

```
SELECT *
FROM Customers
ORDER BY Country DESC;
```

- Order by two columns, one first, and then the next:

```
SELECT *
FROM Customers
ORDER BY Country ASC, CustomerName DESC;
```

LIMIT the number of returned records

- Example to return 5 people from the Persons table:

```
SELECT *
FROM Persons
ORDER BY Last_Name ASC
LIMIT 5;
```

Randomly select and order records

- RAND function can generate random numbers for various uses

- Reorder the entire table:

```
SELECT *
FROM table
ORDER BY RAND();
```

- Randomly select a single record:

```
SELECT *
FROM table
ORDER BY RAND()
LIMIT 1;
```

Randomly select and order records

- Generate a random number in the output results:

```
SELECT id, RAND()
FROM table;
```

Key points from lesson

- The ORDER BY clause specifies that the results from a query should be returned in ascending or descending order
- A LIMIT clause restricts the number of records that would be returned to a subset, which can be convenient for inspection or efficiency
- The RAND() function can be used to generate random values in the output or to randomly sample or randomly order the results of a query

Creating New Tables and Aliases

AS Keyword (Aliases)

- `AS` can be used to create an alias for a field
- A new composite address field can be created from all of the address components as:

```
SELECT CustomerName, CONCAT(Address,', ',
City,', ',PostalCode,', ',Country)
AS Address
FROM Customers;
```

CREATE TABLE AS

- Use CREATE TABLE with AS to create a new table in the database using a select query

```
CREATE TABLE new_table
AS ( SELECT column_name(s)
     FROM old_table);
```

- Matches columns and data types based on the results in the select statement

```
CREATE TABLE CustomersBackup2017
AS ( SELECT CustomerName, ContactName
     FROM Customers);
```

SELECT INTO

- Take the results of a select statement and put them in an existing table or database:

```
SELECT column_name(s)
INTO newtable [IN externaldb]
FROM table1;
```

- Example:

```
SELECT CustomerName, ContactName
INTO CustomersBackup2017
FROM Customers;
```

Key points from lesson

- The AS keyword creates an alias for an attribute or result of a function that is returned in a query

- Results from a query can be inserted into a new table using the CREATE TABLE with the AS keyword

- Results from a query can be inserted into an existing table using a SELECT INTO clause if the table with the appropriate structure already exists

Joining Multiple Tables

Introduction to JOINs

- Last week, our focus was on taking large datasets and normalizing them by separating the data into different tables
- This week, the focus is on taking data from different tables and combining it together
- This may include data from other data sources which complement our own:
  - demographic information for a zip code
  - price structure for shipping zones for a carrier
- The process of merging two separate tables is called "joining"

JOIN details

- Joins effectively merge two tables together based on a relationship between different columns in each table

- Relationships become explicit when you query the database

- Both tables will still contain the normal separated data, and data can still be added to the individual tables

Columns in a JOIN

- Joins may be done on any columns in two tables, as long as the merge operation makes logical sense
  - They don’t need to be keys, though they usually are
  - Join columns must have compatible data types
  - Join column is usually key column: Either primary or foreign
  - NULLs will never join

JOIN Example

- List all orders, showing order number and amount along with the name and credit limit of each customer
  - Orders table has order number and amount, but no customer names or credit limits
  - Customers has customer names and credit limit, but no order info
- Cust = CustNbr

OrderNbr	Cust	Prod	Qty	Amt	Disc
1	211	Bulldozer	7	31000.00	0.20
2	522	Riveter	2	4000.00	0.30
3	522	Crane	1	500000.00	0.40

CustNbr	Company	CustRep	CreditLimit
211	Connor Co	89	50000.00
522	Amaratunga Enterprises	89	40000.00
890	Feni Fabricators	53	1000000.00

JOIN Example

- List all orders, showing order number and amount along with the name and credit limit of each customer

```
SELECT OrderNbr, Amt, Company, CreditLimit
FROM Customers, Orders
WHERE Cust = CustNbr; -- (Implicit syntax)
```

OrderNbr	Cust	Prod	Qty	Amt	Disc
1	211	Bulldozer	7	31000.00	0.20
2	522	Riveter	2	4000.00	0.30
3	522	Crane	1	500000.00	0.40

CustNbr	Company	CustRep	CreditLimit
211	Connor Co	89	50000.00
522	Amaratunga Enterprises	89	40000.00
890	Feni Fabricators	53	1000000.00

JOIN Example

- List all orders, showing order number and amount along with the name and credit limit of each customer

```
SELECT OrderNbr, Amt, Company, CreditLimit
FROM Customers, Orders
WHERE Cust = CustNbr; -- (Implicit syntax)
```

OrderNbr	Amt	Company	CreditLimit
1	31000.00	Connor Co	50000.00
2	4000.00	Amaratunga Enterprises	40000.00
3	500000.00	Amaratunga Enterprises	40000.00

JOIN Example

- List all orders, showing order number and amount along with the name and credit limit of each customer

```
SELECT OrderNbr, Amt, Company, CreditLimit
FROM Customers INNER JOIN Orders
     ON Customers.CustNbr = Orders.Cust;
-- (SQL-92 syntax)
```

OrderNbr	Cust	Prod	Qty	Amt	Disc
1	211	Bulldozer	7	31000.00	0.20
2	522	Riveter	2	4000.00	0.30
3	522	Crane	1	500000.00	0.40

CustNbr	Company	CustRep	CreditLimit
211	Connor Co	89	50000.00
522	Amaratunga Enterprises	89	40000.00
890	Feni Fabricators	53	1000000.00

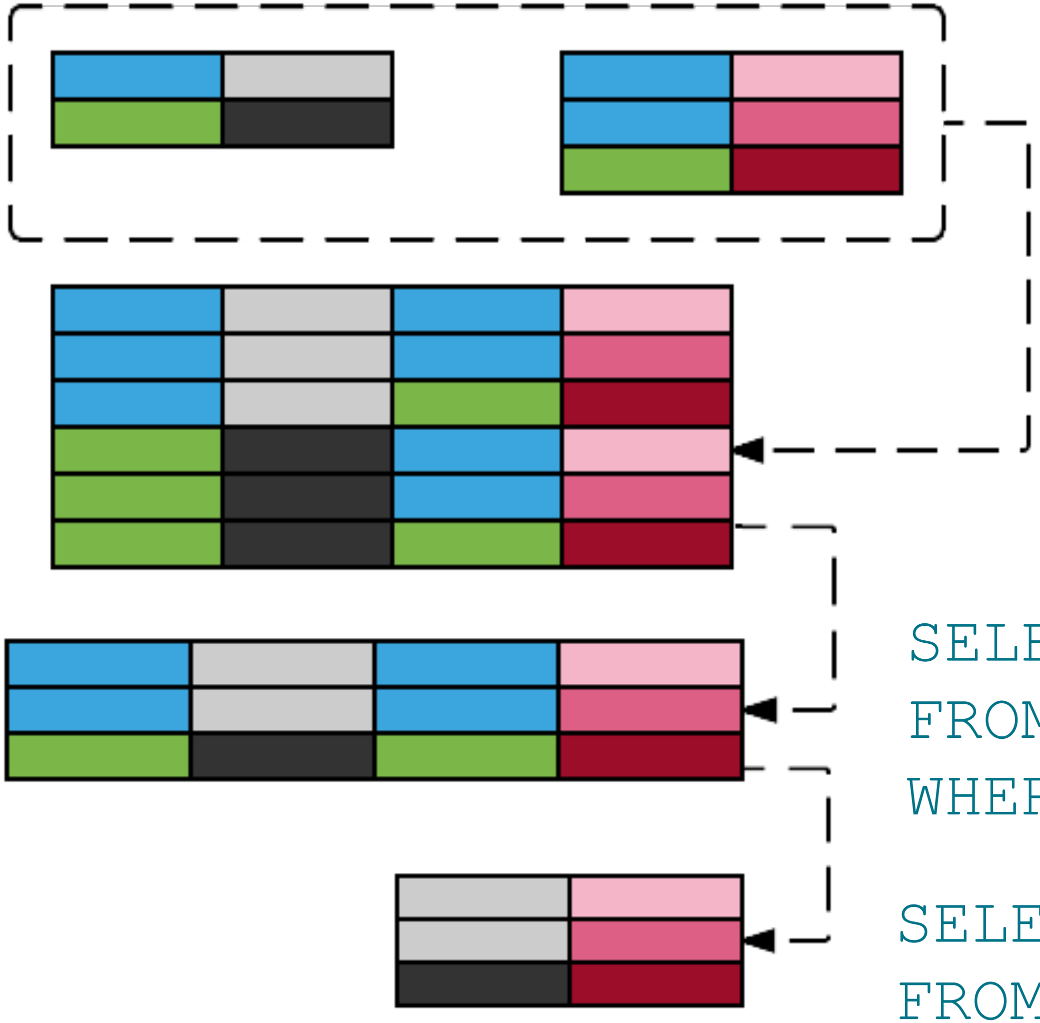
JOIN Example

- List all orders, showing order number and amount along with the name and credit limit of each customer

```
SELECT OrderNbr, Amt, Company, CreditLimit
FROM Customers INNER JOIN Orders
     ON Customers.CustNbr = Orders.Cust;
-- (SQL-92 syntax)
```

OrderNbr	Amt	Company	CreditLimit
1	31000.00	Connor Co	50000.00
2	4000.00	Amaratunga Enterprises	40000.00
3	500000.00	Amaratunga Enterprises	40000.00

Joins - visually



Two tables with shared field/data

```
SELECT *
FROM tb1, tb2;
```

```
SELECT *
FROM tb1, tb2
WHERE tb1.bg = tb2.bg;
```

```
SELECT grey, pink
FROM tb1, tb2
WHERE tb1.bg = tb2.bg;
```

Further notes on JOINs

- Use `*` carefully in joins – it returns all columns from all tables being joined

- If a field has the same name in the tables being joined, specify the table name along with the field name:
  - `table1.fieldname, table2.fieldname`
  - `Customers.CustNbr, Orders.Amt`

Key points from lesson

- The relational database model allows us join multiple tables to build new and unanticipated relationships

- The columns in a join must be of matching types and also must represent the same concept in two different tables

- This can help us to contextualize or integrate a table in our database with data from an external source

Types of JOINs and VIEWs

Join with 3 tables

```
SELECT OrderNbr, Amt, Company, Name
FROM Orders, Customers, SalesReps
WHERE Cust = CustNbr AND CustRep = RepNbr AND
     Amt > 25000;
-- (Implicit syntax)
```

- List orders over $25,000, including the name of the salesperson who took the order and the name of the customer who placed the order

OrderNbr	Cust	Prod	Qty	Amt	Disc
1	211	Bulldozer	7	31000.00	0.20
2	522	Riveter	2	4000.00	0.30
3	522	Crane	1	500000.00	0.40

CustNbr	Company	CustRep	CreditLimit
211	Connor Co	89	50000.00
522	Amaratunga Enterprises	89	40000.00
890	Feni Fabricators	53	1000000.00

RepNbr	Name	RepOffice	Quota	Sales
53	Bill Smith	1	100000.00	0.00
89	Jen Jones	2	50000.00	130000.00

Join with 3 tables

```
SELECT OrderNbr, Amt, Company, Name
FROM Orders, Customers, SalesReps
WHERE Cust = CustNbr AND CustRep = RepNbr AND
      Amt > 25000;
-- (Implicit syntax)
```

- List orders over $25,000, including the name of the salesperson who took the order and the name of the customer who placed the order

OrderNbr	Amt	Company	Names
1	31000.00	Connor Co	Jen Jones
3	500000.00	Amaratunga Enterprises	Jen Jones

Join with 3 tables

```
SELECT OrderNbr, Amt, Company, Name
FROM SalesReps INNER JOIN Customers
      ON SalesReps.RepNbr = Customers.CustRep INNER JOIN Orders
      ON Customers.CustNbr = Orders.Cust
WHERE Amt > 25000;

-- (SQL-92 syntax)
```

- List orders over $25,000, including the name of the salesperson who took the order and the name of the customer who placed the order

OrderNbr	Cust	Prod	Qty	Amt	Disc
1	211	Bulldozer	7	31000.00	0.20
2	522	Riveter	2	4000.00	0.30
3	522	Crane	1	500000.00	0.40

CustNbr	Company	CustRep	CreditLimit
211	Connor Co	89	50000.00
522	Amaratunga Enterprises	89	40000.00
890	Feni Fabricators	53	1000000.00

RepNbr	Name	RepOffice	Quota	Sales
53	Bill Smith	1	100000.00	0.00
89	Jen Jones	2	50000.00	130000.00

Join with 3 tables

```
SELECT OrderNbr, Amt, Company, Name
FROM SalesReps INNER JOIN Customers
       ON SalesReps.RepNbr = Customers.CustRep INNER JOIN Orders
       ON Customers.CustNbr = Orders.Cust
WHERE Amt > 25000;

-- (SQL-92 syntax)
```

- List orders over $25,000, including the name of the salesperson who took the order and the name of the customer who placed the order

OrderNbr	Amt	Company	Names
1	31000.00	Connor Co	Jen Jones
3	500000.00	Amaratunga Enterprises	Jen Jones

JOIN types

```
SELECT OrderNbr, Amt, Company,
CreditLimit FROM Customers, Orders
WHERE Cust = CustNbr;
```

- `INNER JOIN`: returns only the records with matching keys (joins common column values)
- `LEFT JOIN`: returns all rows from `LEFT` (first) table, whether or not they match a record in the second table
- `RIGHT JOIN`: returns all rows from `RIGHT` (second) table, whether or not they match a record in the first table
- `OUTER JOIN`: Returns all rows from both tables, whether or not they match (Microsoft SQL, not MySQL)
- In MySQL, `JOIN` and `INNER JOIN` are equivalent

VIEWs

- VIEWs are virtual tables which present data in a denormalized form to users

- VIEWs DO NOT create separate copies of the data; they reference the data in the underlying tables

- Database stores a definition of a view; the data is updated each time the VIEW is invoked

- Advantages:
  - User queries are simpler on views constructed for them
  - Security: can restrict access to data in views for users
  - Independence: user or program are not affected by small changes in underlying tables

VIEWs

```
CREATE VIEW CustomerOrders AS
SELECT CustNbr, Company, Name, OrderNbr, Prod, Qty, Amt
FROM Customers, SalesReps, Orders
WHERE CustRep = RepNbr AND CustNbr = Cust;
-- (Implicit syntax)
```

OrderNbr	Cust	Prod	Qty	Amt	Disc
1	211	Bulldozer	7	31000.00	0.20
2	522	Riveter	2	4000.00	0.30
3	522	Crane	1	500000.00	0.40

CustNbr	Company	CustRep	CreditLimit
211	Connor Co	89	50000.00
522	Amaratunga Enterprises	89	40000.00
890	Feni Fabricators	53	1000000.00

RepNbr	Name	RepOffice	Quota	Sales
53	Bill Smith	1	100000.00	0.00
89	Jen Jones	2	50000.00	130000.00

VIEWs

```
CREATE VIEW CustomerOrders AS
SELECT CustNbr, Company, Name, OrderNbr, Prod, Qty, Amt
FROM Customers, SalesReps, Orders
WHERE CustRep = RepNbr AND CustNbr = Cust;
-- (Implicit syntax)
```

CustNbr	Company	Name	OrderNbr	Prod	Qty	Amt
211	Connor Co	Jen Jones	1	Bulldozer	7	31000.00
522	Amaratunga Enterprises	Jen Jones	2	Riveter	2	4000.00
522	Amaratunga Enterprises	Jen Jones	3	Crane	1	500000.00

Key points from lesson

- Joining three tables together just involves one additional join between two already joined tables and a third table

- Different types of joins can be used to merge two tables together that always include every row in the left table, right table, or in both tables

- Views are virtual tables which do not change the underlying data but can be helpful to generate reports and simplify complicated queries