1/60 11:47:16 ***

← →

SQL

Where did SQL come from?

As mentioned earlier, SQL is an implementation of Ed Codd's relational set operators:

```
1. SELECT [formerly known as RESTRICT]
2. PROJECT
3. JOIN
4. PRODUCT
5. UNION
6. INTERSECT
7. DIFFERENCE
8. DIVIDE
```

Interestingly, Ed's ideas were ignored by IBM, his employer. Only after Oracle debuted (with SQL support right off the bat!) did IBM create DB2, its first relational DB.

Structured Query Language (SQL)

- Categories of SQL function
  - Data definition language (DDL)
  - Data manipulation language (DML)
- Nonprocedural language with basic command vocabulary set of less than 100 words
- Differences in SQL dialects are minor

now next page (Right Arrow)

Table 7.1 - SQL Data Definition Command

COMMAND OR OPTION	DESCRIPTION
CREATE SCHEMA AUTHORIZATION	Creates a database schema
CREATE TABLE	Creates a new table in the user's database schema
NOT NULL	Ensures that a column will not have null values
UNIQUE	Ensures that a column will not have duplicate values
PRIMARY KEY	Defines a primary key for a table
FOREIGN KEY	Defines a foreign key for a table
DEFAULT	Defines a default value for a column (when no value is given)
CHECK	Validates data in an attribute
CREATE INDEX	Creates an index for a table
CREATE VIEW	Creates a dynamic subset of rows and columns from one or more tables (see Chapter 8, Advanced SQL)
ALTER TABLE	Modifies a table's definition (adds, modifies, or deletes attributes or constraints)
CREATE TABLE AS	Creates a new table based on a query in the user's database schema
DROP TABLE	Permanently deletes a table (and its data)
DROP INDEX	Permanently deletes an index
DROP VIEW	Permanently deletes a view

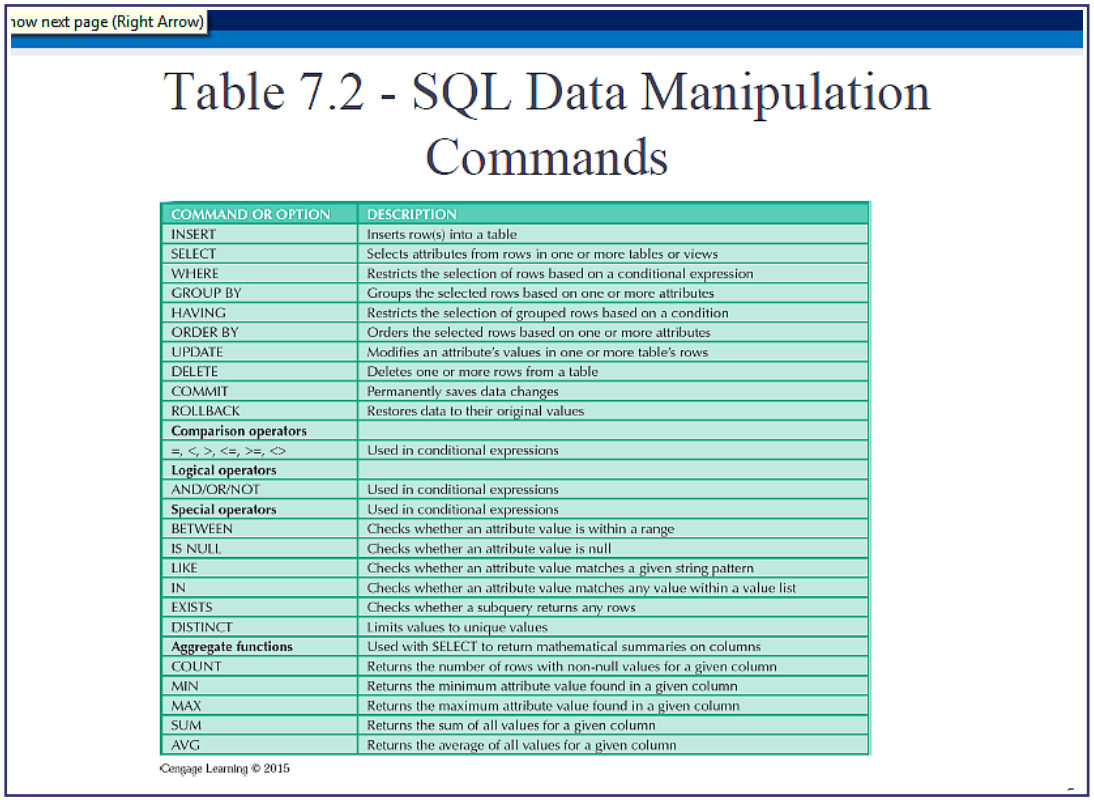
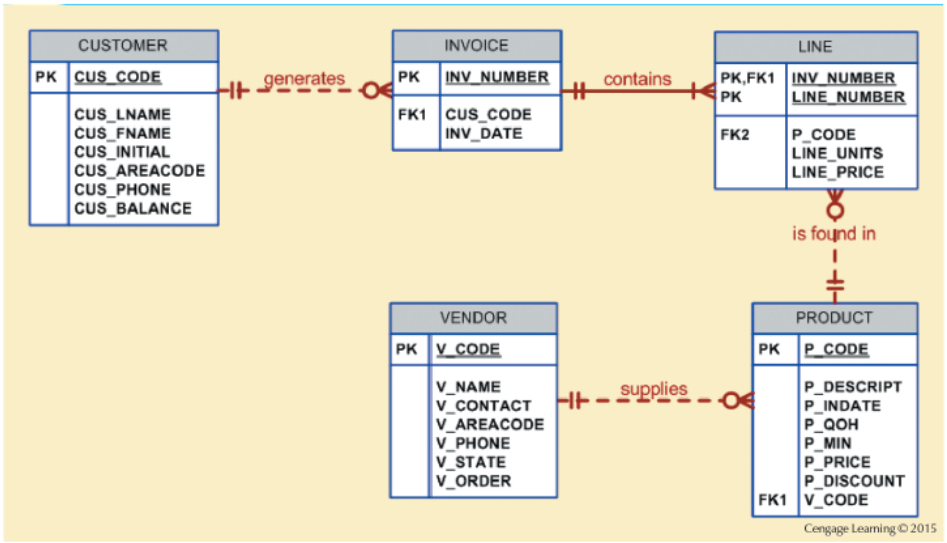
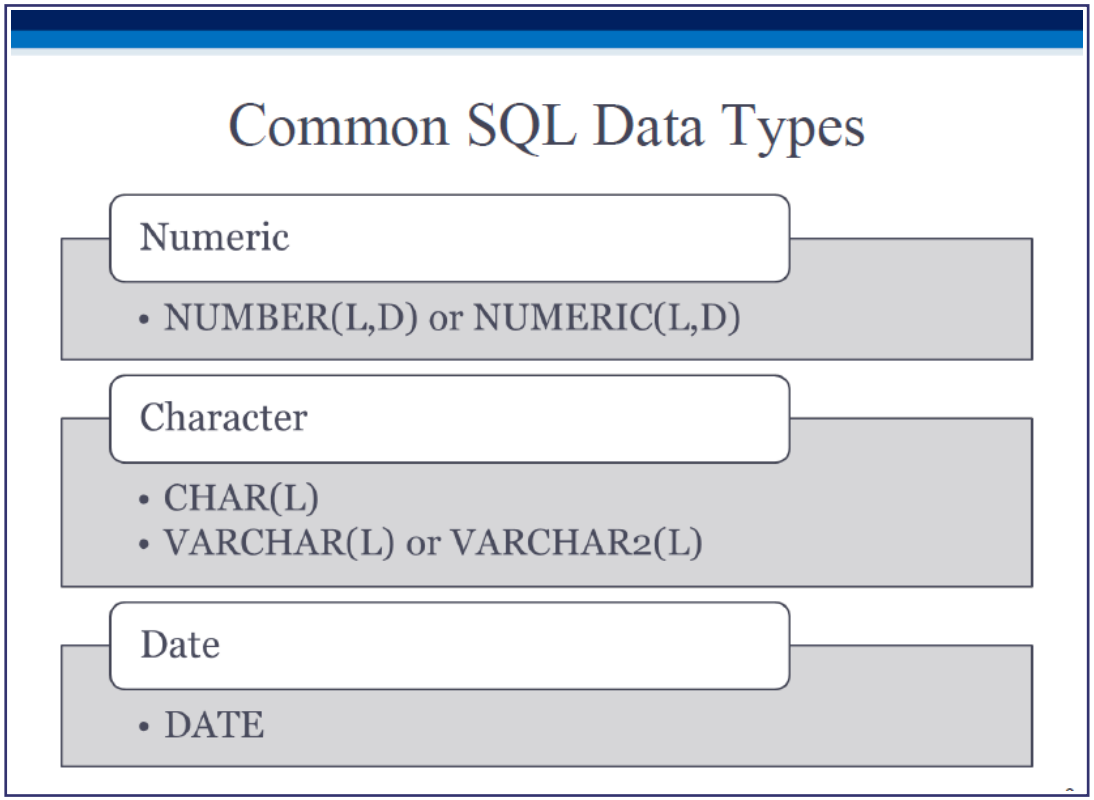


Table 7.2 - SQL Data Manipulation Commands

COMMAND OR OPTION	DESCRIPTION
INSERT	Inserts row(s) into a table
SELECT	Selects attributes from rows in one or more tables or views
WHERE	Restricts the selection of rows based on a conditional expression
GROUP BY	Groups the selected rows based on one or more attributes
HAVING	Restricts the selection of grouped rows based on a condition
ORDER BY	Orders the selected rows based on one or more attributes
UPDATE	Modifies an attribute's values in one or more table's rows
DELETE	Deletes one or more rows from a table
COMMIT	Permanently saves data changes
ROLLBACK	Restores data to their original values
Comparison operators	
=, <, >, <=, >=, <>	Used in conditional expressions
Logical operators	
AND/OR/NOT	Used in conditional expressions
Special operators	Used in conditional expressions
BETWEEN	Checks whether an attribute value is within a range
IS NULL	Checks whether an attribute value is null
LIKE	Checks whether an attribute value matches a given string pattern
IN	Checks whether an attribute value matches any value within a value list
EXISTS	Checks whether a subquery returns any rows
DISTINCT	Limits values to unique values
Aggregate functions	Used with SELECT to return mathematical summaries on columns
COUNT	Returns the number of rows with non-null values for a given column
MIN	Returns the minimum attribute value found in a given column
MAX	Returns the maximum attribute value found in a given column
SUM	Returns the sum of all values for a given column
AVG	Returns the average of all values for a given column

Figure 7.1 - The Database Model





NUMBER(p,s) [or NUMERIC(p,s)] denotes a number (including negative values) with upto 'p' number of total digits, including 's' number of them after the decimal point. 'p' is referred to as precision, 's' denotes scale. Eg. NUMERIC(4,2) would range from -99.99 to 99.99, NUMERIC(3,0) would be -999 to 999.

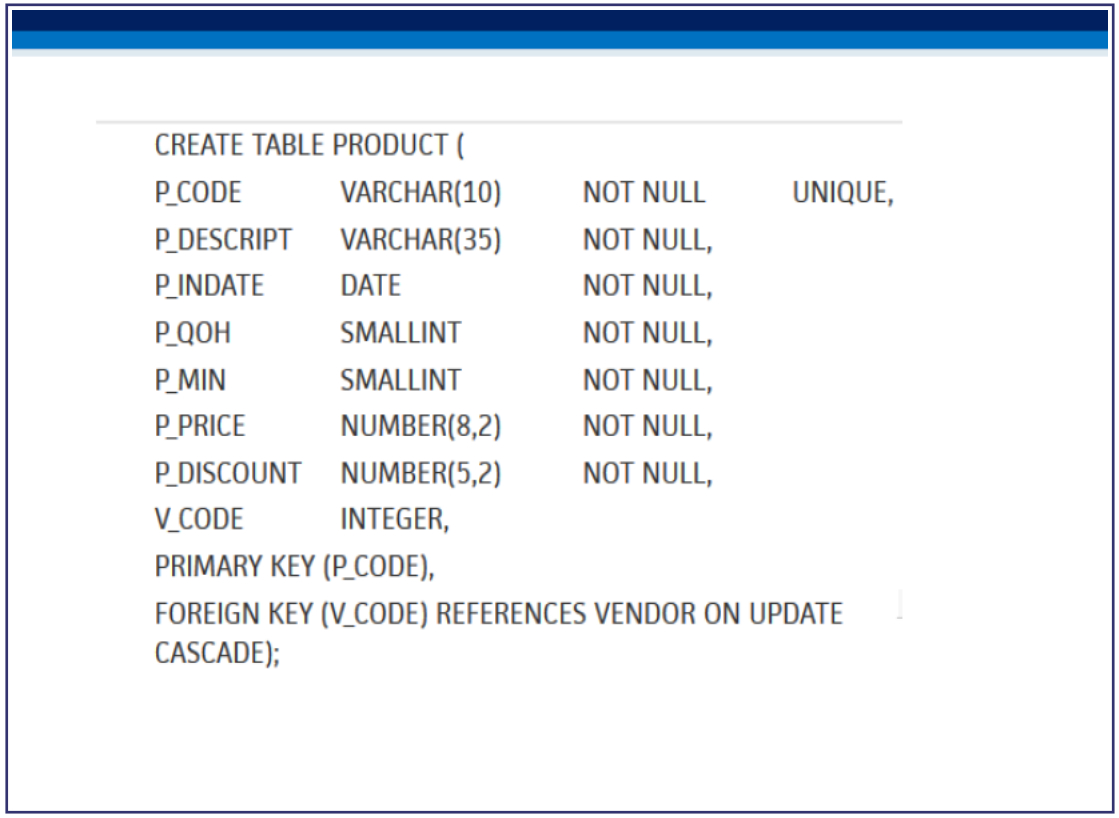
char vs VARCHAR (reserved for future use by SQL - so don't use!) vs VARCHAR2 [versus NVARCHAR2 (for Unicode)]: http://www.orafaq.com/faq/what_is_the_difference_between_varchar_varchar2_and_char_data_types

Creating Table Structures

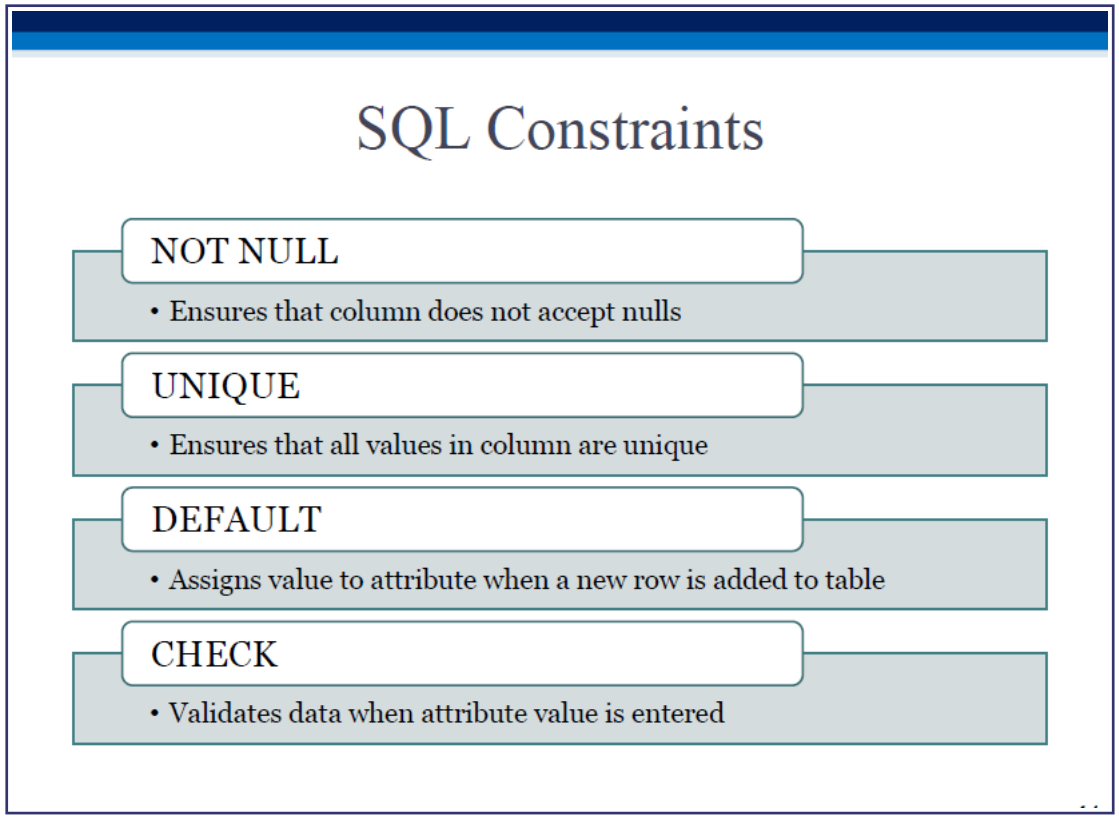
- Use one line per column (attribute) definition
- Use spaces to line up attribute characteristics and constraints
- Table and attribute names are capitalized
- Features of table creating command sequence
  - NOT NULL specification
  - UNIQUE specification
- Syntax to create table
  - CREATE TABLE tablename();

Primary Key and Foreign Key

- Primary key attributes contain both a NOT NULL and a UNIQUE specification
- RDBMS will automatically enforce referential integrity for foreign keys
- Command sequence ends with semicolon
- ANSI SQL allows use of following clauses to cover CASCADE, SET NULL, or SET DEFAULT
  - ON DELETE and ON UPDATE



```
CREATE TABLE PRODUCT (
P_CODE        VARCHAR(10)      NOT NULL      UNIQUE,
P_DESCRIPT    VARCHAR(35)      NOT NULL,
P_INDATE      DATE             NOT NULL,
P_QOH         SMALLINT         NOT NULL,
P_MIN         SMALLINT         NOT NULL,
P_PRICE       NUMBER(8,2)      NOT NULL,
P_DISCOUNT    NUMBER(5,2)      NOT NULL,
V_CODE        INTEGER,
PRIMARY KEY (P_CODE),
FOREIGN KEY (V_CODE) REFERENCES VENDOR ON UPDATE
CASCADE);
```



Plus: ON UPDATE CASCADE and ON DELETE CASCADE - both affect [change] a secondary table (that has an FK), when a change is made in the primary table (with the corresponding PK). ON UPDATE will update the values in the secondary table when corresp. values in the primary table are changed; ON DELETE will delete rows in the secondary table, when linked rows are deleted in the primary table.

Data Manipulation Commands

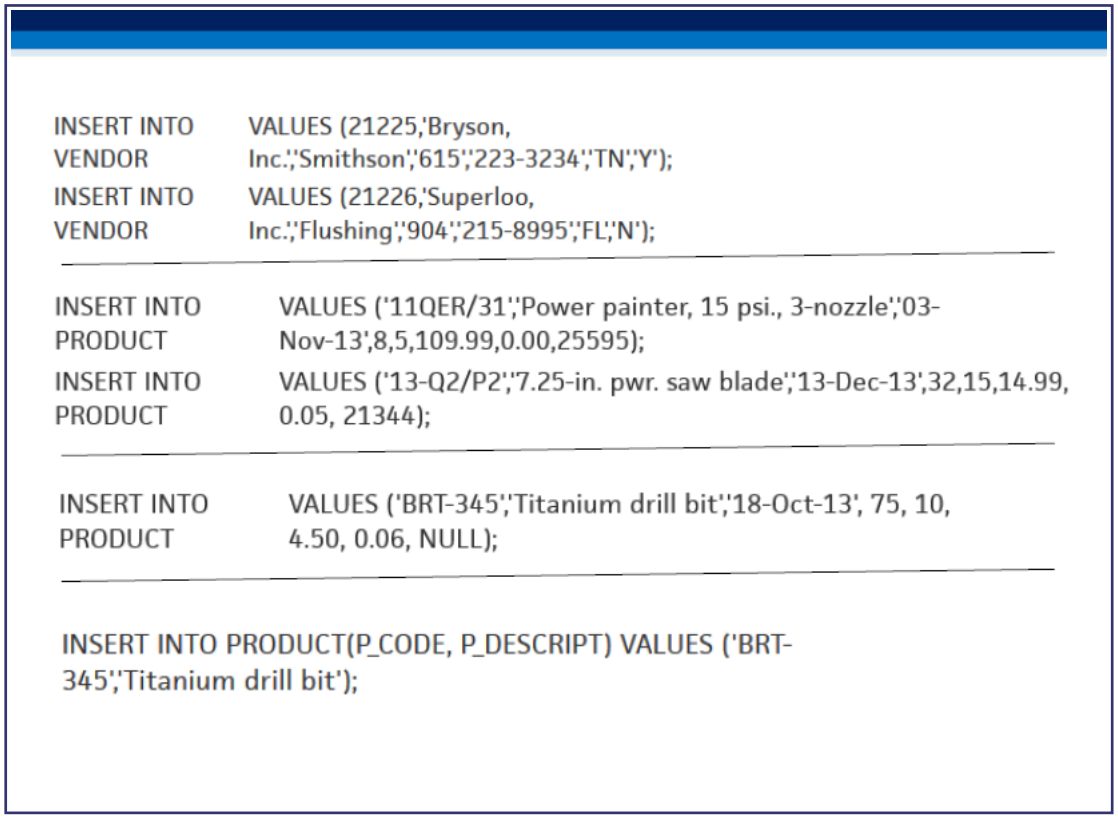
INSERT, SELECT, COMMIT, UPDATE, ROLLBACK, and DELETE.

INSERT: Command to insert data into table

- Syntax - INSERT INTO tablename VALUES();
- Used to add table rows with NULL and NOT NULL attributes

COMMIT: Command to save changes

- Syntax - COMMIT [WORK];
- Ensures database update integrity



```
INSERT INTO VENDOR    VALUES (21225,'Bryson, Inc.','Smithson','615','223-3234','TN','Y');
INSERT INTO VENDOR    VALUES (21226,'Superloo, Inc.','Flushing','904','215-8995','FL','N');

INSERT INTO PRODUCT   VALUES ('11QER/31','Power painter, 15 psi., 3-nozzle','03-Nov-13',8,5,109.99,0.00,25595);
INSERT INTO PRODUCT   VALUES ('13-Q2/P2','7.25-in. pwr. saw blade','13-Dec-13',32,15,14.99, 0.05, 21344);

INSERT INTO PRODUCT   VALUES ('BRT-345','Titanium drill bit','18-Oct-13', 75, 10, 4.50, 0.06, NULL);

INSERT INTO PRODUCT(P_CODE, P_DESCRIPT) VALUES ('BRT-345','Titanium drill bit');
```

Vaguely similar to parameter passing/matching during a function call [where arguments need to match in position, type, count].

Data Manipulation Commands

SELECT: Command to list the contents

- Syntax - SELECT *columnlist* FROM *tablename*;
- **Wildcard character**(*): Substitute for other characters/command

UPDATE: Command to modify data

- Syntax - UPDATE *tablename* SET *columnname* = expression [, *columnname* = *expression*] [WHERE *conditionlist*];

```
SELECT * FROM
PRODUCT;

SELECT    P_CODE, P_DESCRIPT, P_INDATE, P_QOH, P_MIN, P_PRICE, P_DISCOUNT,
          V_CODE
FROM      PRODUCT;

  UPDATE    PRODUCT
  SET       P_INDATE =
            '18-JAN-2014'
  WHERE     P_CODE = '13-Q2/P2';

  UPDATE    PRODUCT
  SET       P_INDATE = '18-JAN-2014', P_PRICE = 17.99, P_MIN
            = 10
  WHERE     P_CODE = '13-Q2/P2';
```

SELECT operates on 1 or more columns, and 0 or more rows from 1 or more tables - like many SQL commands, it is **set-oriented** and **non procedural.**

UPDATE modifies one or more columns of a table (on all rows, or on specific rows based on a condition specified by WHERE). Here is more.

Data Manipulation Commands

WHERE condition

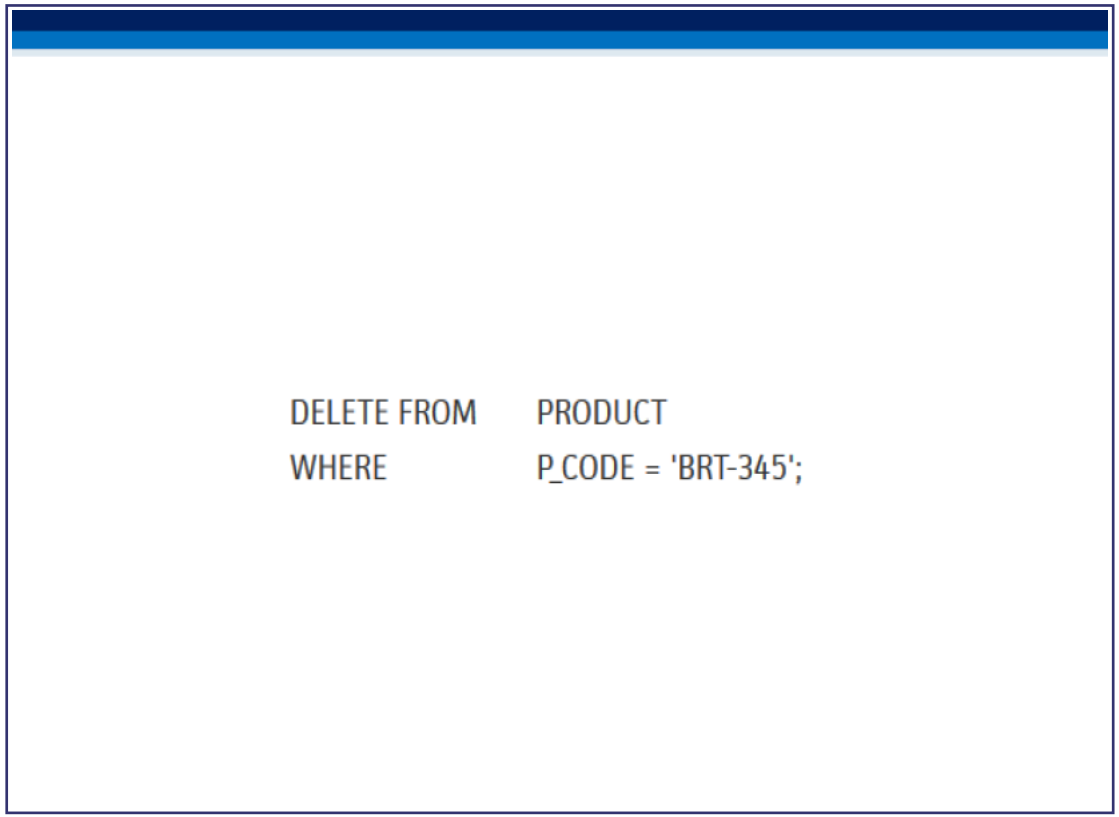
- Specifies the rows to be selected

ROLLBACK: Command to restore the database

- Syntax - ROLLBACK;
- Undoes the changes since last COMMIT command

DELETE: Command to delete

- Syntax - DELETE FROM *tablename*
  - [WHERE *conditionlist*];

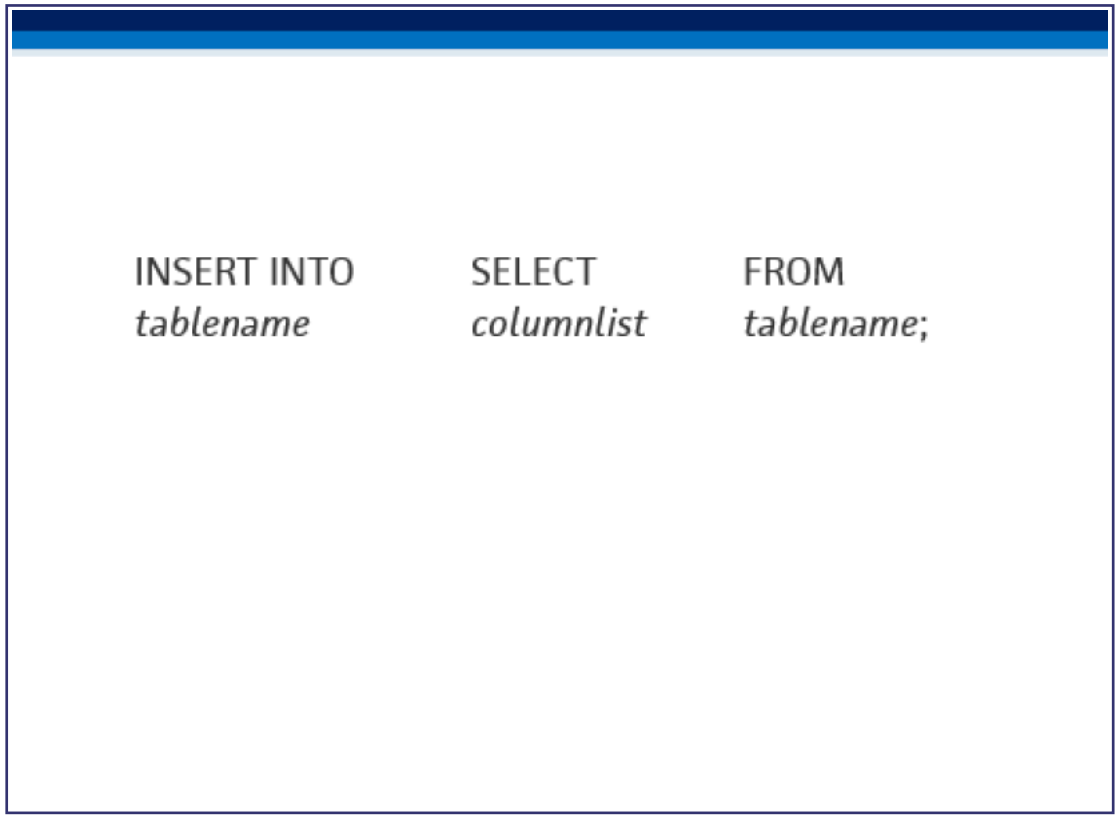


Another ex: DELETE FROM PRODUCT WHERE P_MIN=5; (can be any condition on any attribute).

Q: what would DELETE do, if there isn't a WHERE condition?

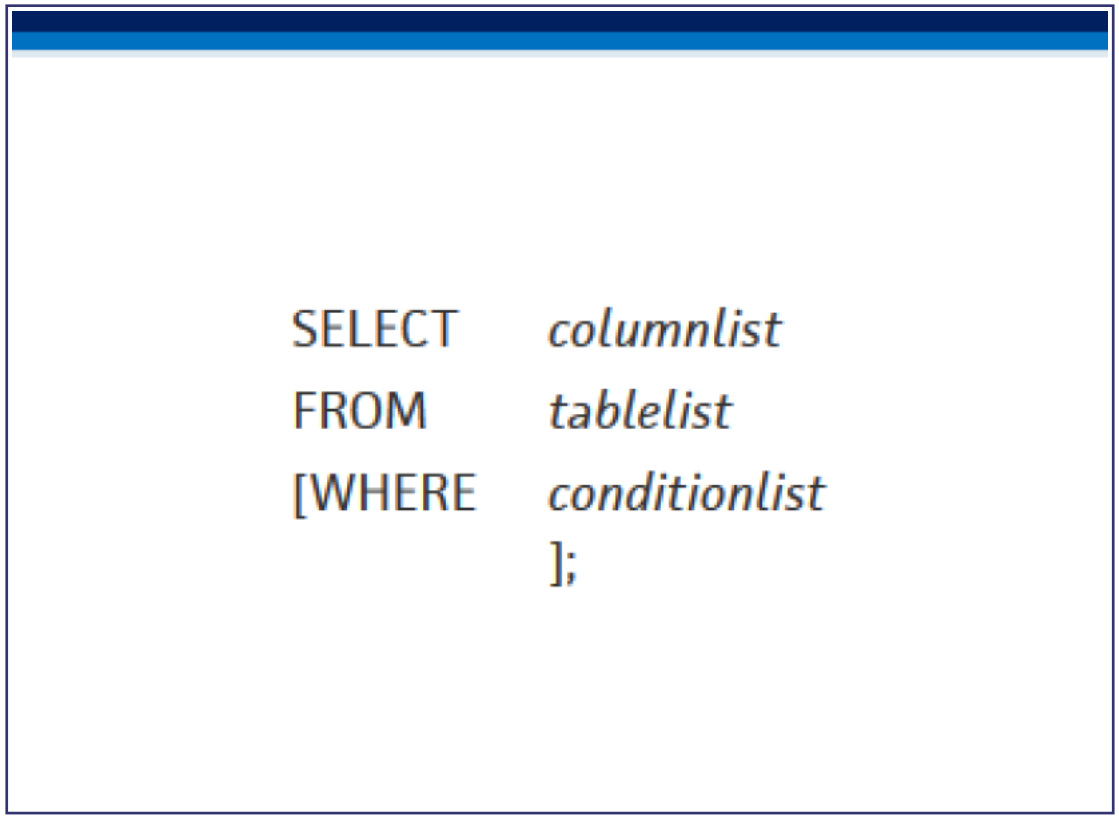
Inserting Table Rows with a SELECT Subquery

- Syntax
  - INSERT INTO *tablename* SELECT *columnlist* FROM *tablename*
- Used to add multiple rows using another table as source
- SELECT command - Acts as a subquery and is executed first
  - **Subquery**: Query embedded/nested inside another query


INSERT INTO
tablename
SELECT
columnlist
FROM
tablename;

Selecting Rows Using Conditional Restrictions

- Following syntax enables to specify which rows to select
  - SELECT *columnlist*
  - FROM *tablelist*
  - [WHERE *conditionlist*];
- Used to select partial table contents by placing restrictions on the rows
- Optional WHERE clause
  - Adds conditional restrictions to the SELECT statement


SELECT columnlist
FROM tablelist
[WHERE conditionlist
];

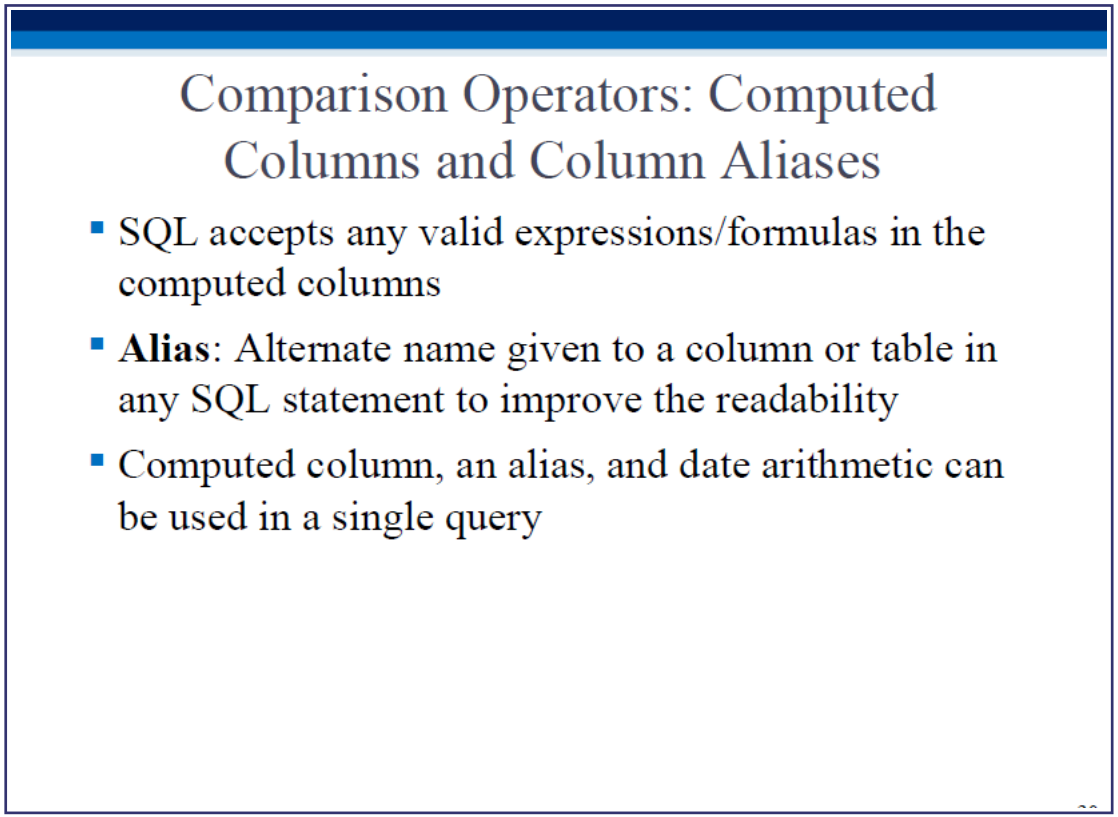
Comparison Operators

- Add conditional restrictions on selected table contents
- Used on:
  - Character attributes
  - Dates

Table 7.6 - Comparison Operators

SYMBOL	MEANING
=	Equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
<> or !=	Not equal to

Computed columns (an ex of 'feature engineering')




```
SELECT    P_DESCRIPT, P_QOH, P_PRICE, P_QOH *
          P_PRICE
FROM      PRODUCT
```

P_DESCRIPT	P_QOH	P_PRICE	Expr1
Power painter, 15 psi., 3-nozzle	8	109.99	879.92
7.25-in. pwr. saw blade	32	14.99	479.68
9.00-in. pwr. saw blade	18	17.49	314.82
Hrd. cloth, 1/4-in., 2x50	15	39.95	599.25
Hrd. cloth, 1/2-in., 3x50	23	43.99	1011.77
B&D jigsaw, 12-in. blade	8	109.92	879.36
B&D jigsaw, 8-in. blade	6	99.87	599.22
B&D cordless drill, 1/2-in.	12	38.95	467.40
Claw hammer	23	9.95	228.85
Sledge hammer, 12 lb.	8	14.40	115.20
Rat-tail file, 1/8-in. fine	43	4.99	214.57
Hicut chain saw, 16 in.	11	256.99	2826.89
PVC pipe, 3.5-in., 8-ft	188	5.87	1103.56
1.25-in. metal screw, 25	172	6.99	1202.28
2.5-in. wd. screw, 50	237	8.45	2002.65
Steel matting, 4'x8'x1/6", .5" mesh	18	119.95	2159.10

```
SELECT    P_DESCRIPT, P_QOH, P_PRICE, P_QOH * P_PRICE AS TOTVALUE
FROM      PRODUCT;
```

P_DESCRIPT	P_QOH	P_PRICE	TOTVALUE
Power painter, 15 psi., 3-nozzle	8	109.99	879.92
7.25-in. pwr. saw blade	32	14.99	479.68
9.00-in. pwr. saw blade	18	17.49	314.82
Hrd. cloth, 1/4-in., 2x50	15	39.95	599.25
Hrd. cloth, 1/2-in., 3x50	23	43.99	1011.77
B&D jigsaw, 12-in. blade	8	109.92	879.36
B&D jigsaw, 8-in. blade	6	99.87	599.22
B&D cordless drill, 1/2-in.	12	38.95	467.40
Claw hammer	23	9.95	228.85
Sledge hammer, 12 lb.	8	14.40	115.20
Rat-tail file, 1/8-in. fine	43	4.99	214.57
Hicut chain saw, 16 in.	11	256.99	2826.89
PVC pipe, 3.5-in., 8-ft	188	5.87	1103.56
1.25-in. metal screw, 25	172	6.99	1202.28
2.5-in. wd. screw, 50	237	8.45	2002.65
Steel matting, 4'x8'x1/6", .5" mesh	18	119.95	2159.10

```
SELECT    P_CODE, P_INDATE, P_INDATE + 90 AS EXPDATE
FROM      PRODUCT;
```

Arithmetic operators

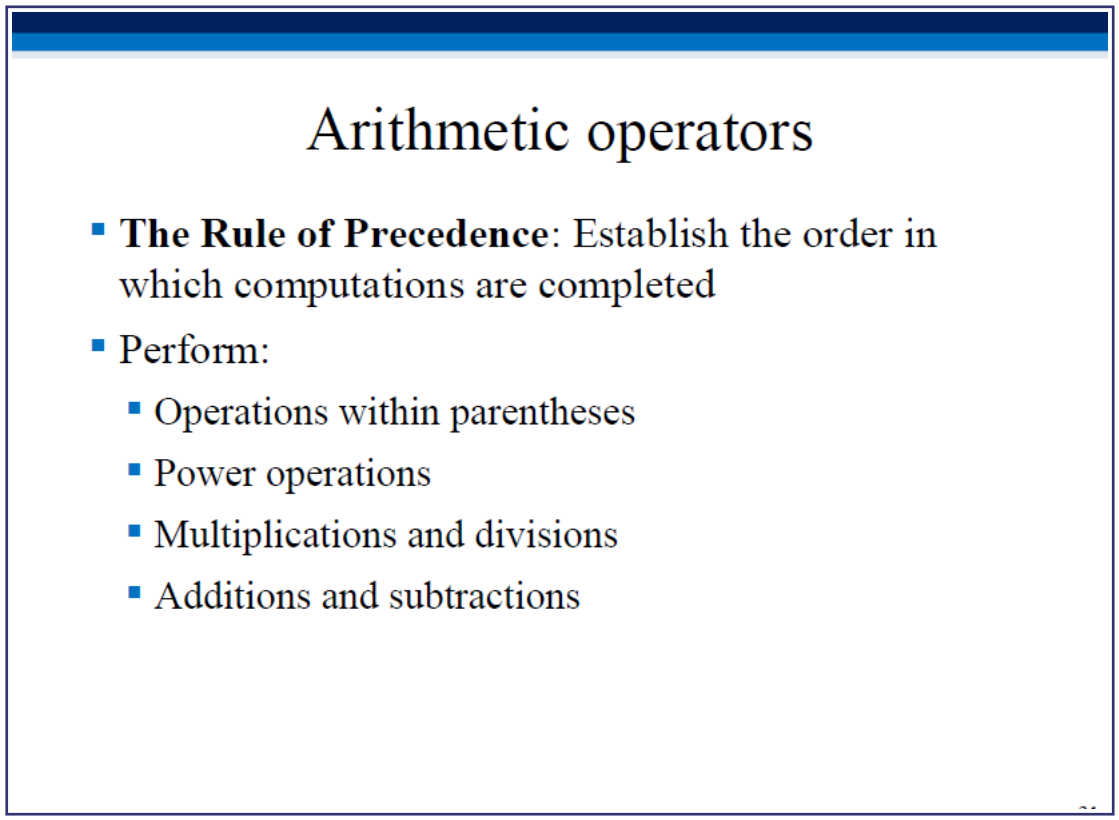


Table 7.7 - The Arithmetic Operators

ARITHMETIC OPERATOR	DESCRIPTION
+	Add
-	Subtract
*	Multiply
/	Divide
^	Raise to the power of (some applications use ** instead of ^)

Logical (Boolean) operators

Figure 7.12 - Selected PRODUCT Table Attributes: The logical OR

P_DESCRIPT	P_INDATE	P_PRICE	V_CODE
7.25-in. pwr. saw blade	13-Dec-13	14.99	21344
9.00-in. pwr. saw blade	13-Nov-13	17.49	21344
B&D jigsaw, 12-in. blade	30-Dec-13	109.92	24288
B&D jigsaw, 8-in. blade	24-Dec-13	99.87	24288
Rat-tail file, 1/8-in. fine	15-Dec-13	4.99	21344
Hicut chain saw, 16 in.	07-Feb-14	256.99	24288


```
SELECT P_DESCRIPT, P_INDATE, P_PRICE, V_CODE
FROM   PRODUCT
WHERE  V_CODE = 21344 OR V_CODE = 24288;
```

Figure 7.13 - Selected PRODUCT Table Attributes: The Logical AND

P_DESCRIPT	P_INDATE	P_PRICE	V_CODE
B&D cordless drill, 1/2-in.	20-Jan-14	38.95	25595
Claw hammer	20-Jan-14	9.95	21225
PVC pipe, 3.5-in., 8-ft	20-Feb-14	5.87	
1.25-in. metal screw, 25	01-Mar-14	6.99	21225
2.5-in. wd. screw, 50	24-Feb-14	8.45	21231

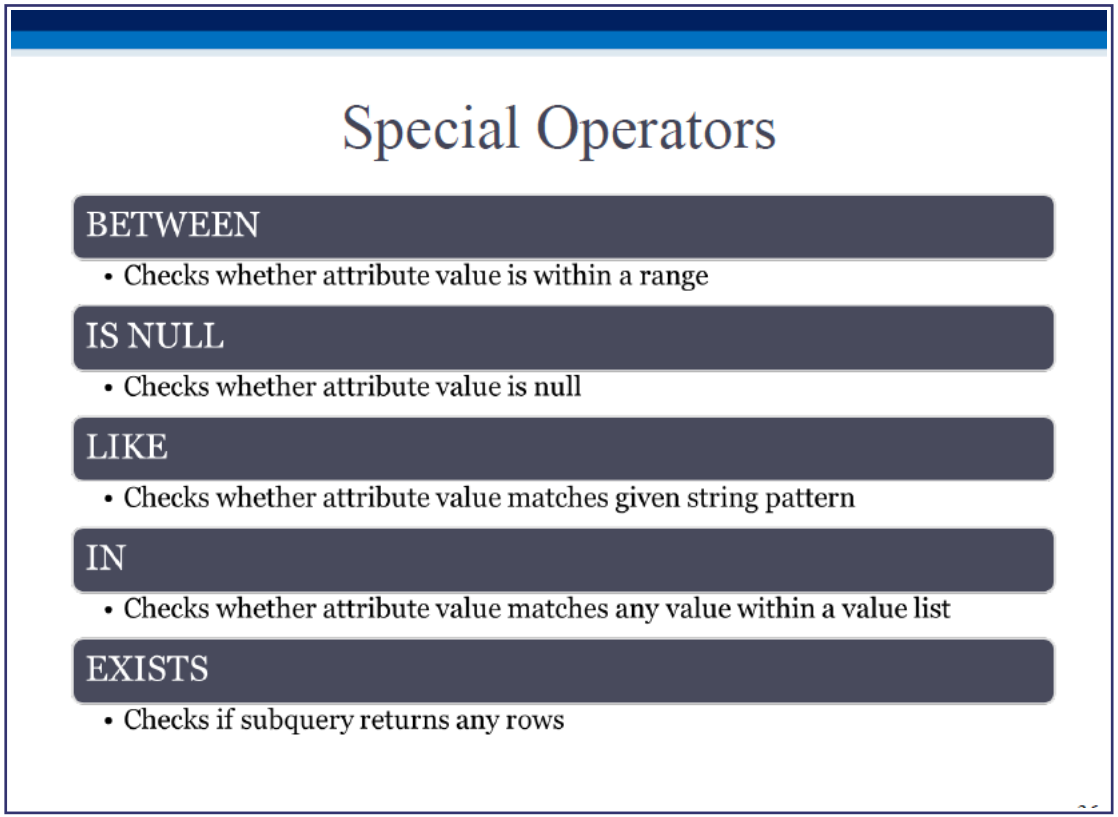

```
SELECT    P_DESCRIPT, P_INDATE, P_PRICE, V_CODE
FROM      PRODUCT
WHERE     P_PRICE < 50
AND       P_INDATE > '15-Jan-2014';
```

Figure 7.14 - Selected PRODUCT Table Attributes: The Logical AND and OR

P_DESCRIPT	P_INDATE	P_PRICE	V_CODE
B&D jigsaw, 12-in. blade	30-Dec-13	109.92	24288
B&D jigsaw, 8-in. blade	24-Dec-13	99.87	24288
B&D cordless drill, 1/2-in.	20-Jan-14	38.95	25595
Claw hammer	20-Jan-14	9.95	21225
Hicut chain saw, 16 in.	07-Feb-14	256.99	24288
PVC pipe, 3.5-in., 8-ft	20-Feb-14	5.87	
1.25-in. metal screw, 25	01-Mar-14	6.99	21225
2.5-in. wd. screw, 50	24-Feb-14	8.45	21231


```
SELECT    P_DESCRIPT, P_INDATE, P_PRICE, V_CODE
FROM      PRODUCT
WHERE     (P_PRICE < 50 AND P_INDATE > '15-Jan-2014')
OR        V_CODE = 24288;
```

'Special' operators



BETWEEN

```
SELECT      *
FROM        PRODUCT
WHERE       P_PRICE BETWEEN 50.00 AND 100.00;
```

```
SELECT      P_CODE, P_DESCRIPT, V_CODE
FROM        PRODUCT
WHERE       V_CODE IS NULL;
```

LIKE

- % means any and all *following* or *preceding* characters are eligible. For example:

  'J%' includes Johnson, Jones, Jernigan, July, and J-231Q.

  'Jo%' includes Johnson and Jones.

  '%n' includes Johnson and Jernigan.

- _means any *one* character may be substituted for the underscore. For example:

  '_23-456-6789' includes 123-456-6789, 223-456-6789, and 323-456-6789.

  '_23-_56-678_' includes 123-156-6781, 123-256-6782, and 823-956-6788.

  '_o_es' includes Jones, Cones, Cokes, totes, and roles.

```
SELECT    V_NAME, V_CONTACT, V_AREACODE, V_PHONE
FROM      VENDOR
WHERE     V_CONTACT LIKE 'Smith%';
```

Another example for pattern-matching: load this page, and enter and execute:

```
SELECT * FROM Customers WHERE CustomerName LIKE 'C%';
```

IN

```
SELECT      *
FROM        PRODUCT
WHERE       V_CODE = 21344
OR          V_CODE = 24288;

SELECT      *
FROM        PRODUCT
WHERE       V_CODE IN (21344, 24288);


SELECT      V_CODE, V_NAME
FROM        VENDOR
WHERE       V_CODE IN (SELECT V_CODE FROM PRODUCT);
```

'EXISTS'

The EXISTS clause is used with a query, and returns TRUE if the subquery results in any output (non-zero # of rows being returned), or FALSE if the subquery results in no data. The rest of the query (the 'main' query) will (or will not) run, based on EXIST's output - if EXISTS returns false, the main query will get skipped.

You can loosely think of EXISTS as "ONLY WHEN". We use it to 'defensively' update (insert, modify, delete) parts of a table (after we determine it is updatable).

You can also think of 'WHERE EXISTS' as "such that there exists". Eg. in the first example below (SELECT * FROM VENDOR..), it reads as "Find all vendors such that there exists records for them in the PRODUCT TABLE (via V_CODE) where P_QOH<=P_MIN" [in other words, we are looking for vendors we need to re-order from].

In the second example (INSERT INTO CONTACTS..), read it as "get the supplier ID and name for all suppliers such that there exists order IDs for them, then insert them into the contacts table".

Exercise: what does #3 below (the UPDATE query) do?

```
SELECT      *
FROM        VENDOR
WHERE       EXISTS (SELECT * FROM PRODUCT WHERE P_QOH <= P_MIN);
```

```
INSERT INTO contacts
(contact_id, contact_name)
SELECT supplier_id, supplier_name
FROM suppliers
WHERE EXISTS (SELECT *
              FROM orders
              WHERE suppliers.supplier_id = orders.supplier_id);
```

```
UPDATE suppliers
SET supplier_name = (SELECT customers.name
                     FROM customers
                     WHERE customers.customer_id = suppliers.supplier_id)
WHERE EXISTS (SELECT customers.name
              FROM customers
              WHERE customers.customer_id = suppliers.supplier_id);
```

```
DELETE FROM suppliers
WHERE EXISTS (SELECT *
              FROM orders
              WHERE suppliers.supplier_id = orders.supplier_id);
```

A couple more (ex of EXISTS, NOT EXISTS):

```
SELECT *
FROM suppliers
WHERE EXISTS (SELECT *
              FROM orders
              WHERE suppliers.supplier_id = orders.supplier_id);
```

This SQL EXISTS condition example will return all records from the suppliers table where there is at least one record in the orders table with the same supplier_id.

```
SELECT *
FROM suppliers
WHERE NOT EXISTS (SELECT *
                  FROM orders
                  WHERE suppliers.supplier_id = orders.supplier_id);
```

This SQL EXISTS example will return all records from the suppliers table where there are **no** records in the *orders* table for the given supplier_id.

ALTER

Advanced Data Definition Commands

- **ALTER TABLE** command: To make changes in the table structure
- Keywords use with the command
  - ADD - Adds a column
  - MODIFY - Changes column characteristics
  - DROP - Deletes a column
- Used to:
  - Add table constraints
  - Remove table constraints

Changing Column's Data Type

- ALTER can be used to change data type
- Some RDBMSs do not permit changes to data types unless column is empty
- Syntax –
  - ALTER TABLE *tablename* MODIFY *(columnname(datatype))*;

```
ALTER TABLE PRODUCT   MODIFY (V_CODE CHAR(5));
```

Changing Column's Data Characteristics

- Use ALTER to change data characteristics
- Changes in column's characteristics are permitted if changes do not alter the existing data type
- Syntax
  - ALTER TABLE *tablename* MODIFY *(columnname(characterstic))*;

```
ALTER TABLE PRODUCT   MODIFY (P_PRICE DECIMAL(9,2));
```

Adding Column, Dropping Column

- Adding a column
  - Use ALTER and ADD
  - Do not include the NOT NULL clause for new column
- Dropping a column
  - Use ALTER and DROP
  - Some RDBMSs impose restrictions on the deletion of an attribute

```
ALTER TABLE PRODUCT   ADD (P_SALECODE CHAR(1));

ALTER TABLE VENDOR   DROP COLUMN V_ORDER;
```

The UPDATE command

Advanced Data Updates

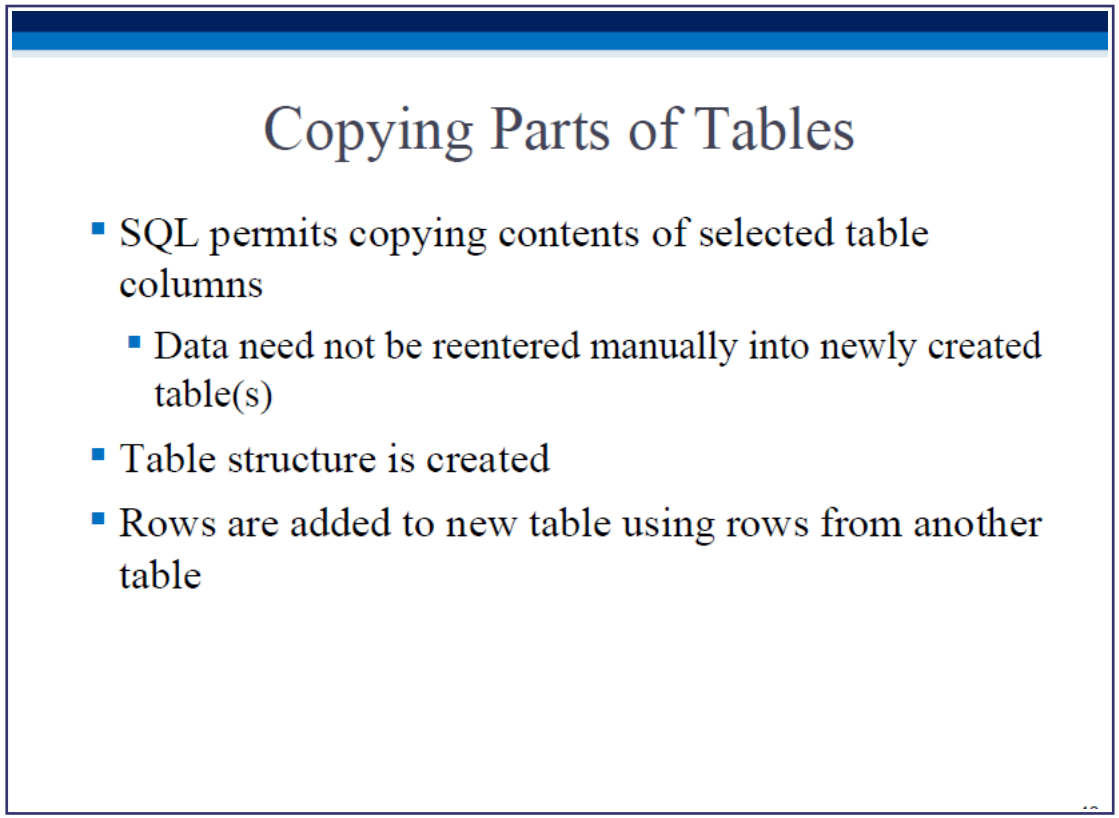
- UPDATE command updates only data in existing rows
- If a relationship is established between entries and existing columns, the relationship can assign values to appropriate slots
- Arithmetic operators are useful in data updates
- In Oracle, ROLLBACK command undoes changes made by last two UPDATE statements

```
UPDATE      PRODUCT
SET         P_SALECODE = '2'
WHERE       P_CODE = '1546-QQ2';

UPDATE      PRODUCT
SET         P_SALECODE = '1'
WHERE       P_CODE IN ('2232/QWE', '2232/QTY');

UPDATE      PRODUCT
SET         P_SALECODE = '1'
WHERE       P_INDATE >= '16-Jan-2014' AND P_INDATE <='10-Feb-2014';

UPDATE      PRODUCT
SET         P_PRICE = P_PRICE * 1.10
WHERE       P_PRICE < 50.00;
```

Table update in 'bulk'

Sometimes we can update a table, by filling it with output from a query. The table to be filled in has to exist first (so we need to create it if necessary), and have type-compatible columns that can receive values from our data-fetching query.

The table creation and updating can happen in separate steps, or even be combined for compactness of expression.



```
CREATE TABLE PART(
PART_CODE                 CHAR(8),
PART_DESCRIPT             CHAR(35),
PART_PRICE                DECIMAL(8,2),
V_CODE                    INTEGER,
PRIMARY KEY (PART_CODE));

INSERT INTO PART          (PART_CODE, PART_DESCRIPT, PART_PRICE, V_CODE)
SELECT                    P_CODE, P_DESCRIPT, P_PRICE, V_CODE FROM PRODUCT;
```

```
CREATE TABLE PART AS
SELECT    P_CODE AS PART_CODE, P_DESCRIPT AS
          PART_DESCRIPT, P_PRICE AS PART_ PRICE,
          V_CODE
FROM      PRODUCT;
```

Adding Primary and Foreign Key Designations

- **ALTER TABLE** command
  - Followed by a keyword that produces the specific change one wants to make
  - Options include ADD, MODIFY, and DROP
- Syntax to add or modify columns
  - ALTER TABLE *tablename*
    - {ADD | MODIFY} (*columnname datatype* [{ADD | MODIFY} *columnname datatype*]) ;
  - ALTER TABLE *tablename*
    - ADD *constraint* [ADD *constraint*] ;

```
ALTER TABLE   PART
    ADD       PRIMARY KEY (PART_CODE);


ALTER TABLE   PART
    ADD       FOREIGN KEY (V_CODE) REFERENCES VENDOR;
```

Oftentimes the need to do this occurs when a table (eg. PART) is created via bulk-update, from another table (eg. PRODUCT).

Deleting a Table from the Database

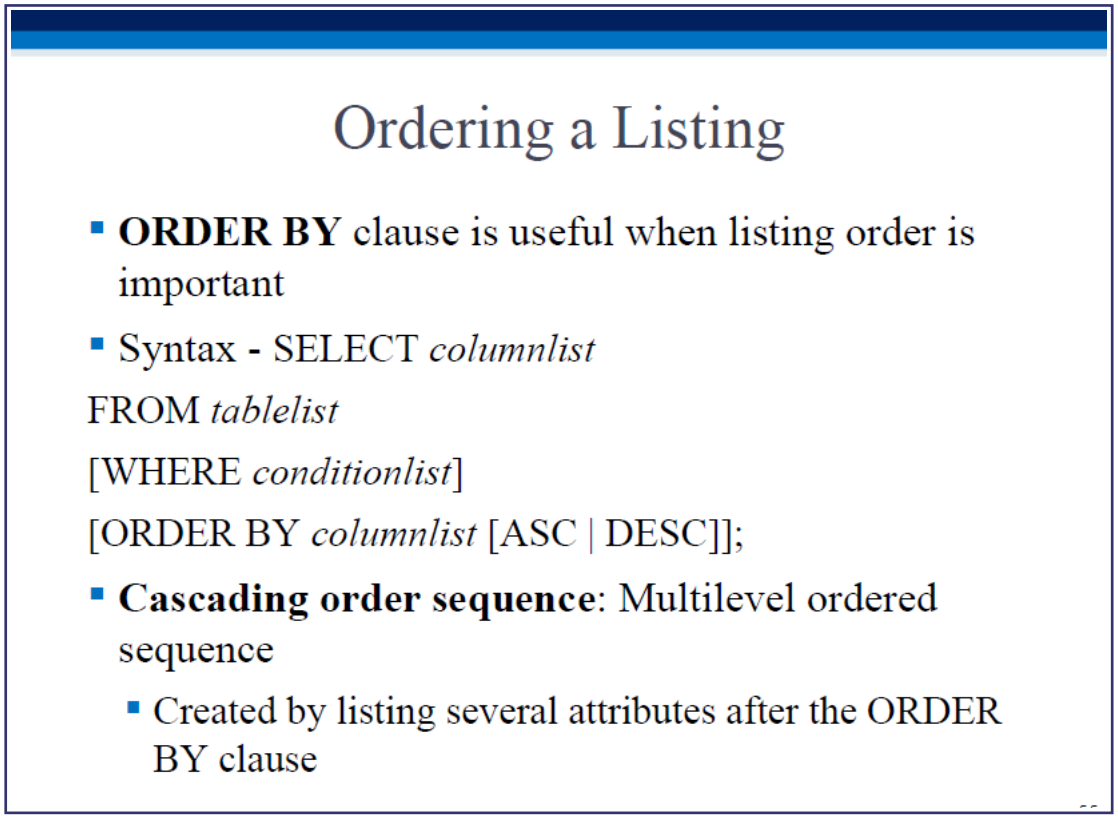
- **DROP TABLE**: Deletes table from database
- Syntax - DROP TABLE tablename;
- Can drop a table only if it is not the one side of any relationship
- RDBMS generates a foreign key integrity violation error message if the table is dropped

Ordering, unique entries, aggregate ops..

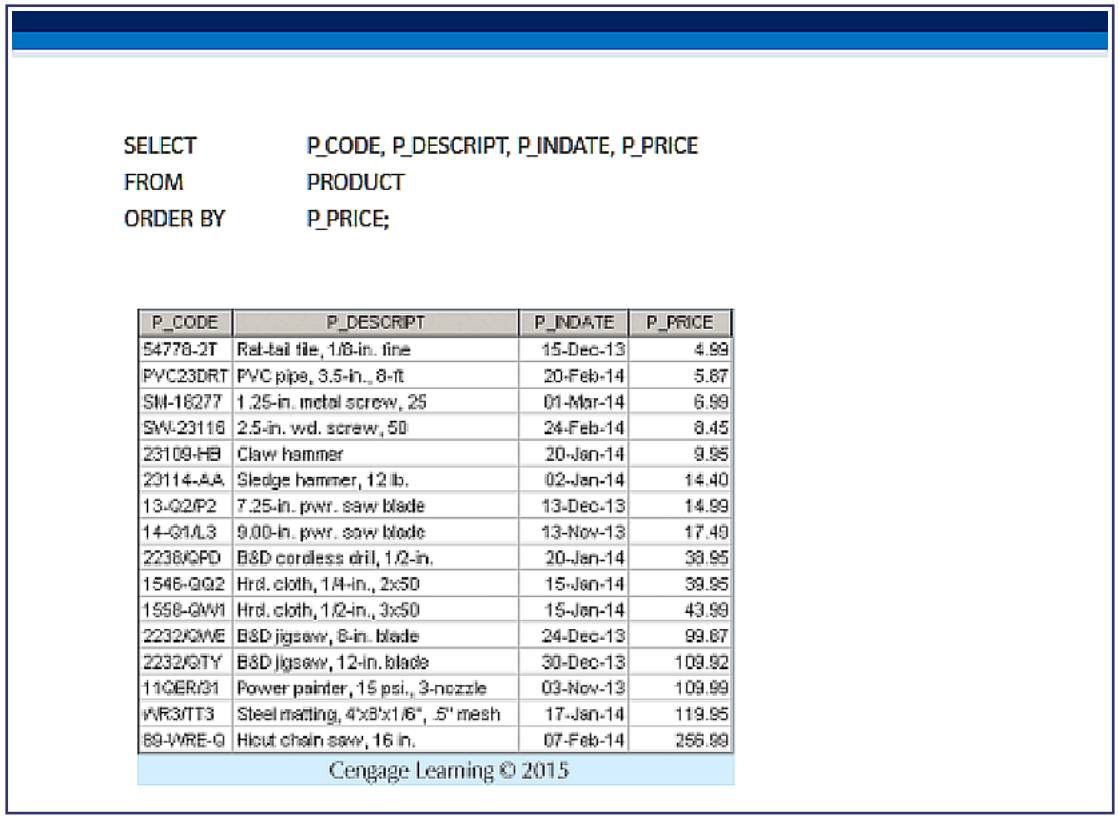
Additional SELECT Query Keywords

- Logical operators work well in the query environment
- SQL provides useful functions that:
  - Counts
  - Find minimum and maximum values
  - Calculate averages
- SQL allows user to limit queries to entries:
  - Having no duplicates
  - Whose duplicates may be grouped

ORDER BY



Below is an example of ORDER BY, where by default, values are listed (sorted) in ascending order. To list the values in descending order, we'd do this: ORDER BY P_PRICE DESC;



```
SELECT      P_CODE, P_DESCRIPT, P_INDATE, P_PRICE
FROM        PRODUCT
ORDER BY    P_PRICE;
```

P_CODE	P_DESCRIPT	P_INDATE	P_PRICE
54778-2T	Rat-tail file, 1/8-in. fine	15-Dec-13	4.99
PVC23DRT	PVC pipe, 3.5-in., 8-ft	20-Feb-14	5.87
SM-16277	1.25-in. metal screw, 25	01-Mar-14	6.99
SW-23116	2.5-in. wd. screw, 50	24-Feb-14	8.45
23109-HB	Claw hammer	20-Jan-14	9.95
23114-AA	Sledge hammer, 12 lb.	02-Jan-14	14.40
13-Q2/P2	7.25-in. pwr. saw blade	13-Dec-13	14.99
14-Q1/L3	9.00-in. pwr. saw blade	13-Nov-13	17.49
2238/QPD	B&D cordless drill, 1/2-in.	20-Jan-14	38.95
1546-QQ2	Hrd. cloth, 1/4-in., 2x50	15-Jan-14	39.95
1558-QW1	Hrd. cloth, 1/2-in., 3x50	15-Jan-14	43.99
2232/QWE	B&D jigsaw, 8-in. blade	24-Dec-13	99.87
2232/QTY	B&D jigsaw, 12-in. blade	30-Dec-13	109.92
11QER/31	Power painter, 15 psi., 3-nozzle	03-Nov-13	109.99
WR3/TT3	Steel matting, 4'x8'x1/6", .5" mesh	17-Jan-14	119.95
89-WRE-Q	Hicut chain saw, 16 in.	07-Feb-14	256.99

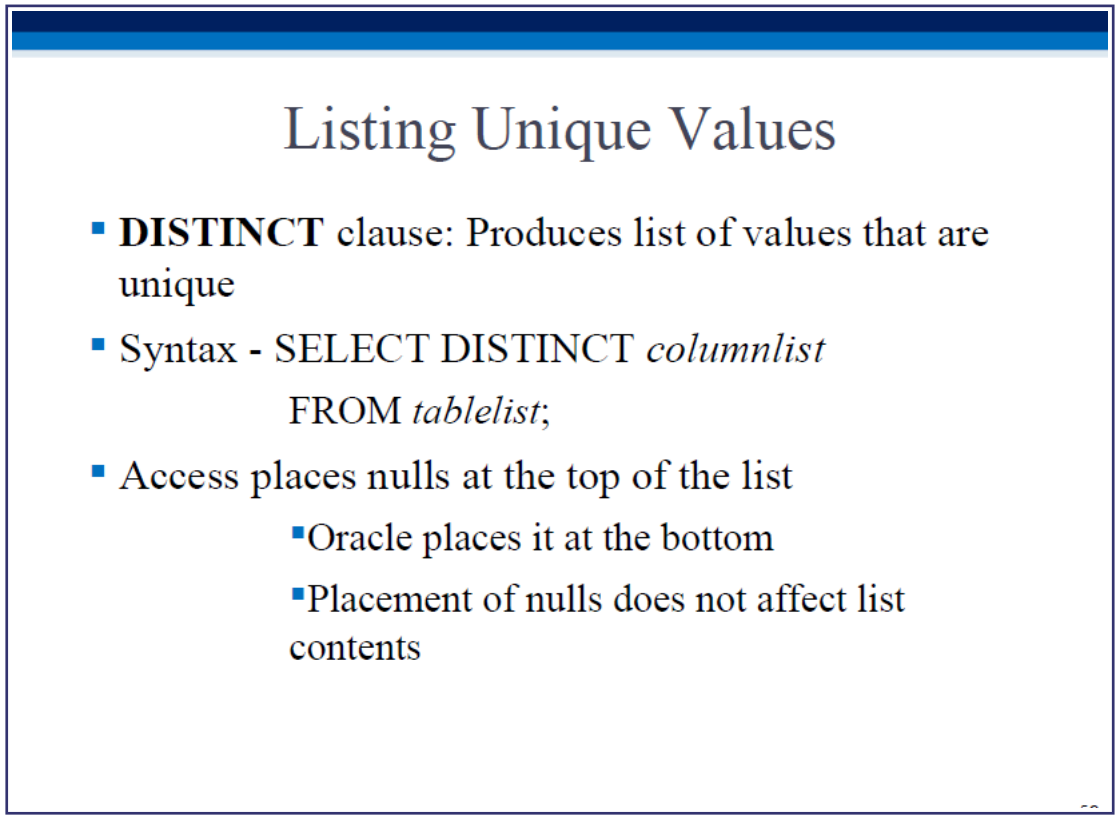
The sequence (listing) obtained by specifying several comma-separated attributes for ORDER BY is called a cascading order sequence.

```
SELECT      EMP_LNAME, EMP_FNAME, EMP_INITIAL, EMP_AREACODE, EMP_PHONE
FROM        EMPLOYEE
ORDER BY    EMP_LNAME, EMP_FNAME, EMP_INITIAL;
```

EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_AREACODE	EMP_PHONE
Brandon	Marie	G	901	882-0845
Diante	Jorge	D	615	890-4567
Genkazi	Leighla	W	901	569-0093
Johnson	Edward	E	615	898-4387
Jones	Anne	M	615	898-3456
Kolmycz	George	D	615	324-5456
Lange	John	P	901	504-4430
Lewis	Rhonda	G	615	324-4472
Saranda	Hermine	R	615	324-5505
Smith	George	A	615	890-2984
Smith	George	K	901	504-3339
Smith	Jeanine	K	615	324-7883
Smythe	Melanie	P	615	324-9006
Vandam	Rhett		901	675-8993
Washington	Rupert	E	615	890-4925
Wiesenbach	Paul	R	615	897-4358
Williams	Robert	D	615	890-3220

DISTINCT

The 'DISTINCT' keyword is used to count unique/distinct occurences of an attribute:



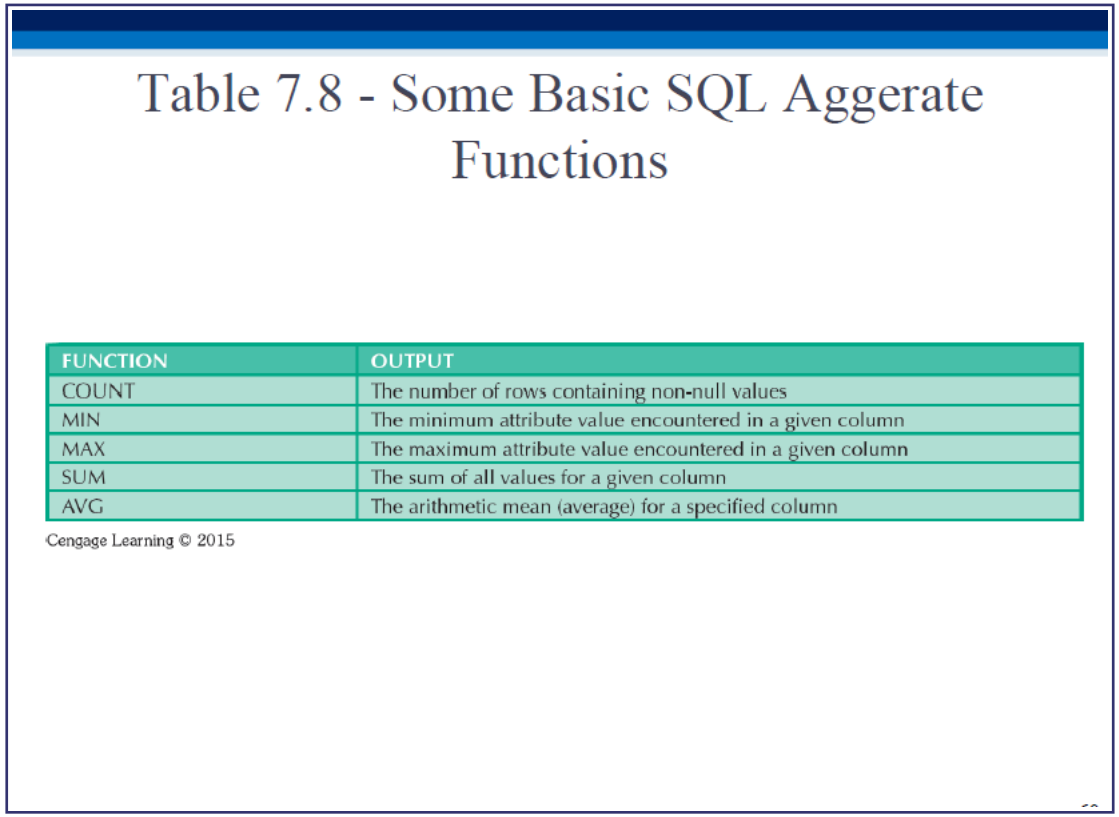
Example of 'DISTINCT' usage

```
SELECT    DISTINCT V_CODE
FROM      PRODUCT;
```

V_CODE
21225
21231
21344
23119
24288
25595

AGGREGATE functions

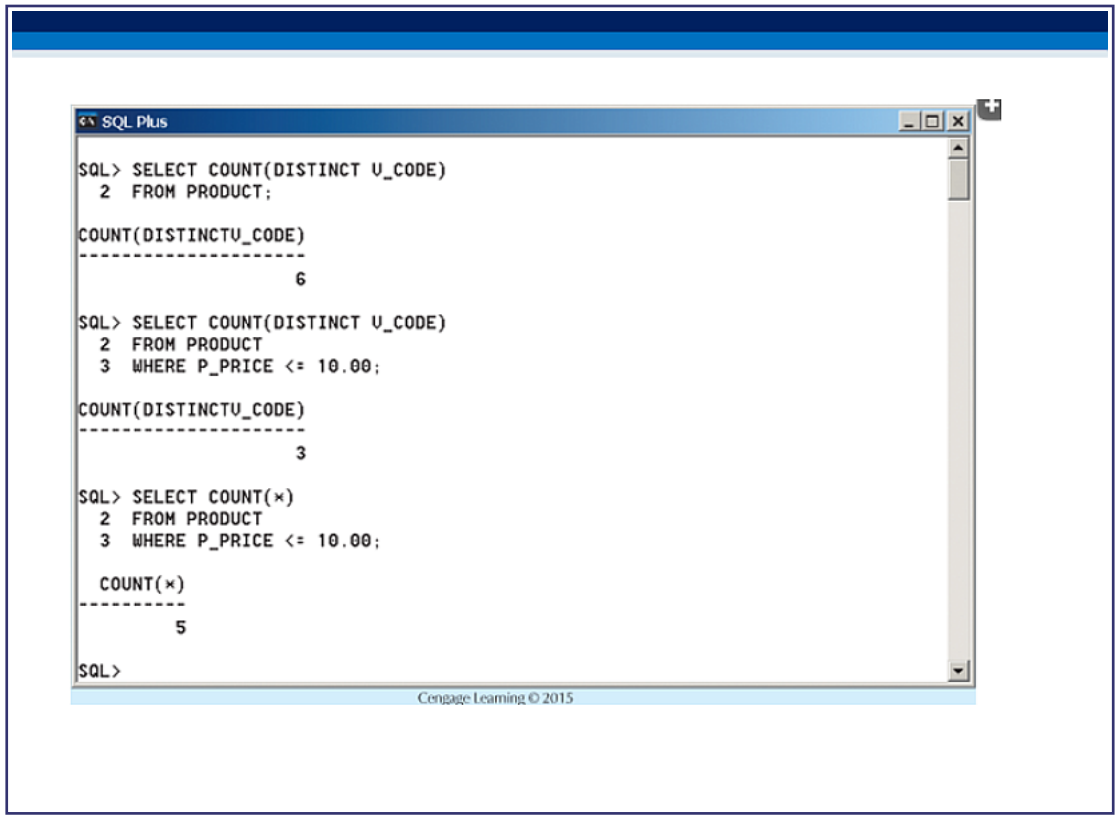
COUNT, MIN, MAX, SUM, AVG are all functions that operate on a numerical attr/column, and produce a single scalar result (not a table).



Aggregate function examples

How many different vendors supply our products? How many supply cheap (PRICE < 10) products?

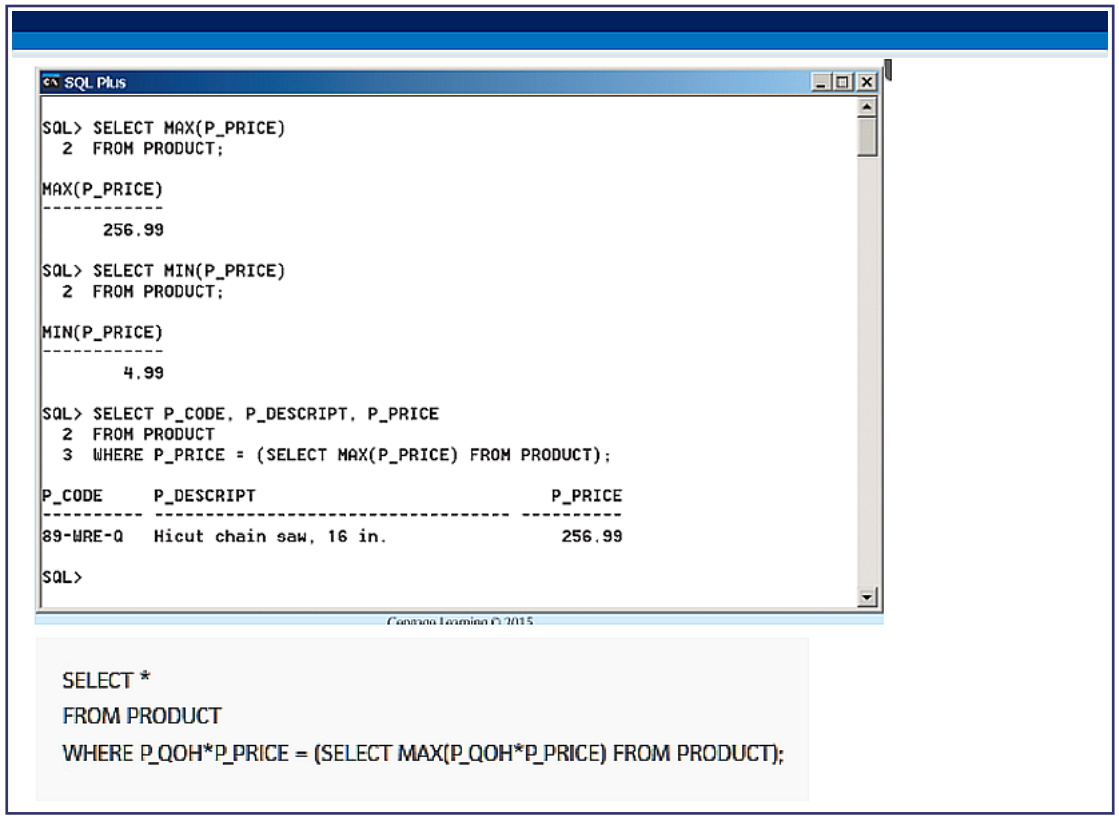
Note the third query below: COUNT(*) counts the # of rows returned by a query (rows where the product costs < 10 units). In contrast, count() counts the # of non-null values of the column.



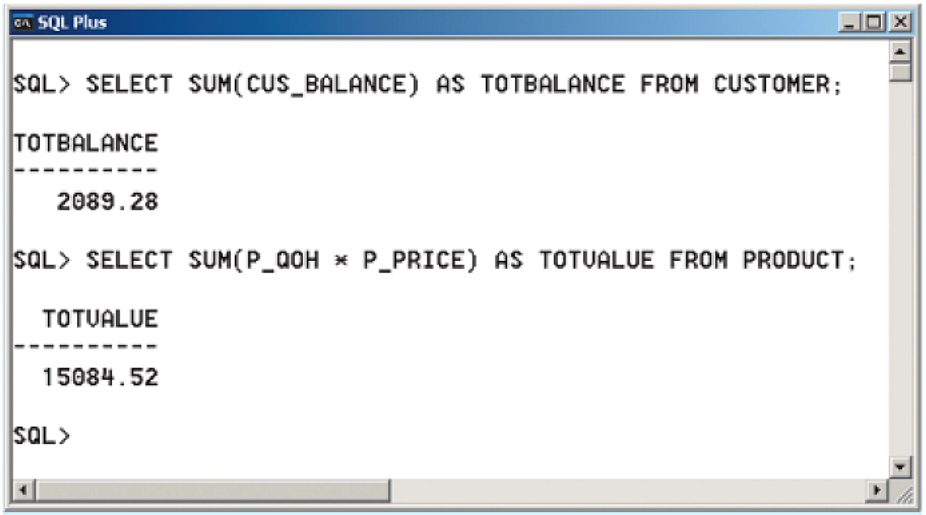
MAX, MIN

What is the value for the most expensive item in the product table? Least expensive?

What is the most expensive item (details)? We can't do: WHERE P_PRICE = MAX(P_PRICE). This is because MAX() can only be used in a SELECT statement.



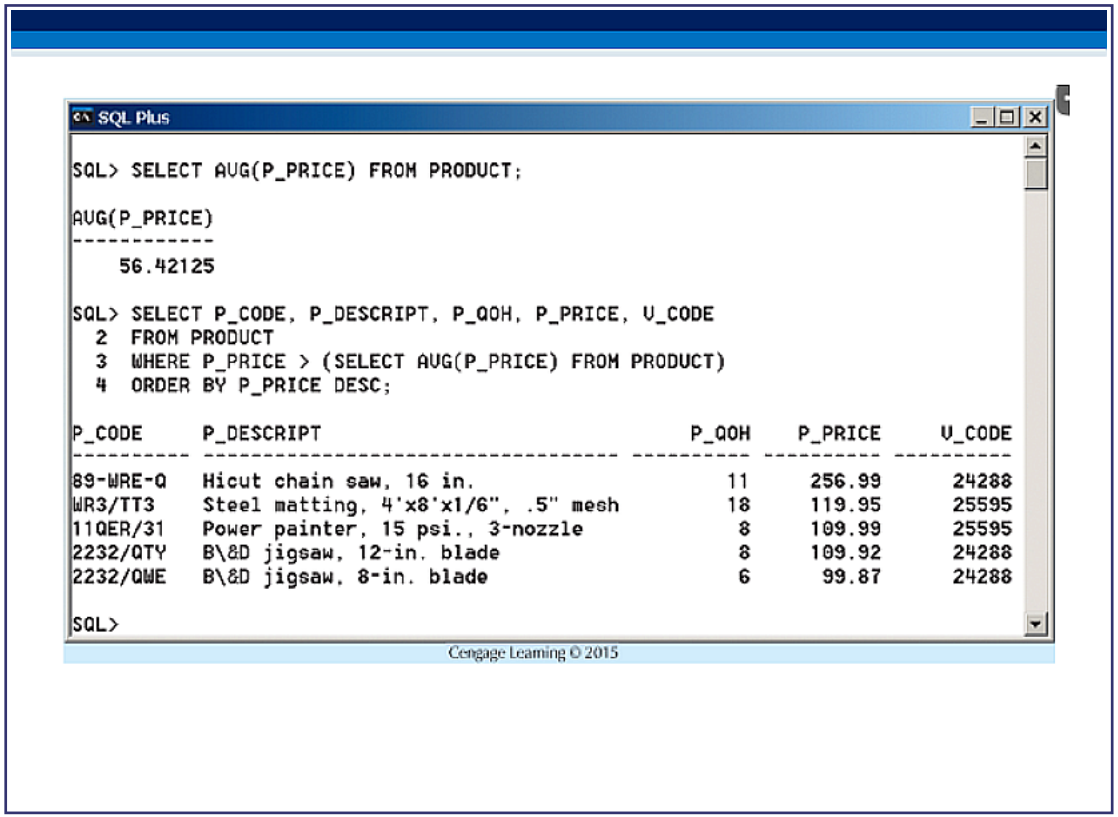
SUM


SQL Plus
SQL> SELECT SUM(CUS_BALANCE) AS TOTBALANCE FROM CUSTOMER;
TOTBALANCE

2089.28
SQL> SELECT SUM(P_QOH * P_PRICE) AS TOTVALUE FROM PRODUCT;
TOTVALUE

15084.52
SQL>
Cengage Learning © 2015

AVG


SQL Plus
SQL> SELECT AVG(P_PRICE) FROM PRODUCT;
AVG(P_PRICE)
56.42125
SQL> SELECT P_CODE, P_DESCRIPT, P_QOH, P_PRICE, V_CODE
2 FROM PRODUCT
3 WHERE P_PRICE > (SELECT AVG(P_PRICE) FROM PRODUCT)
4 ORDER BY P_PRICE DESC;
P_CODE P_DESCRIPT P_QOH P_PRICE V_CODE
89-WRE-Q Hicut chain saw, 16 in. 11 256.99 24288
WR3/TT3 Steel matting, 4'x8'x1/6", .5" mesh 18 119.95 25595
11QER/31 Power painter, 15 psi., 3-nozzle 8 109.99 25595
2232/QTY B\&D jigsaw, 12-in. blade 8 109.92 24288
2232/QWE B\&D jigsaw, 8-in. blade 6 99.87 24288
SQL>
Cengage Learning © 2015

GROUP BY (itemizing)

Grouping Data

- Frequency distributions created by **GROUP BY** clause within SELECT statement
- GROUP BY can **only** be used in concert with an aggregate function
- Syntax - SELECT *columnlist*

  FROM *tablelist*

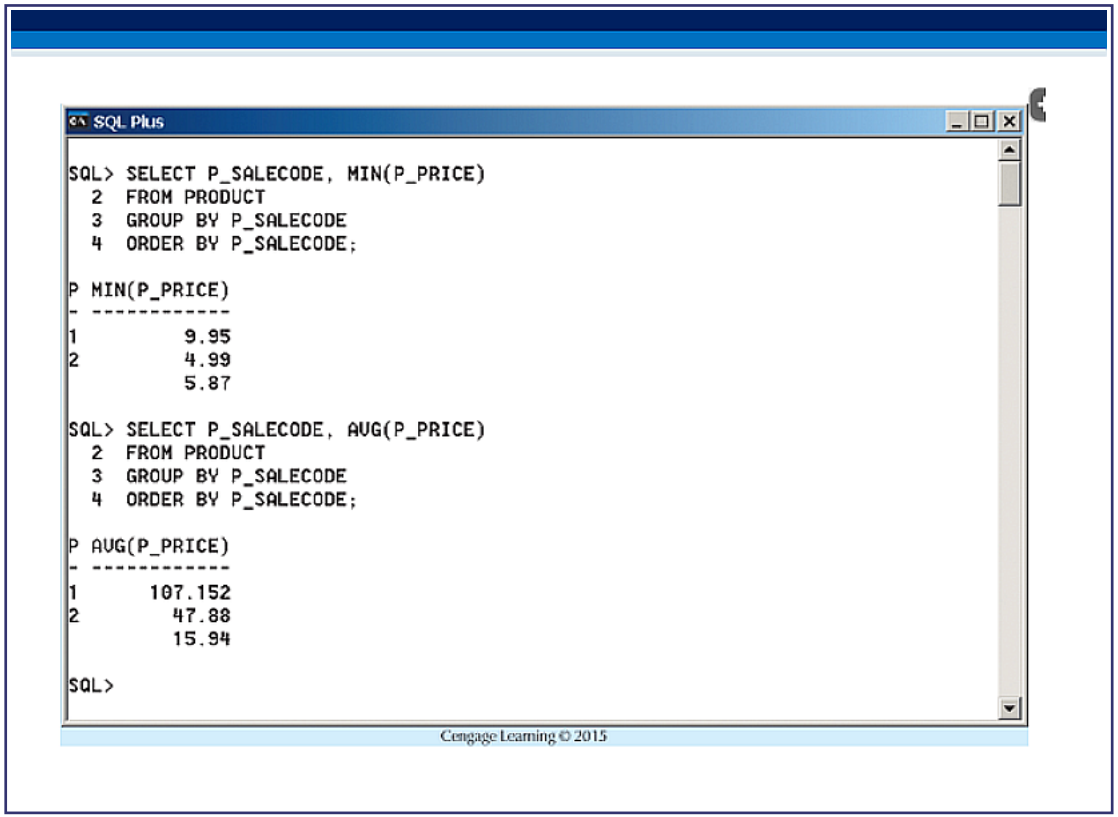
  [WHERE *conditionlist*]

  [GROUP BY *columnlist*]

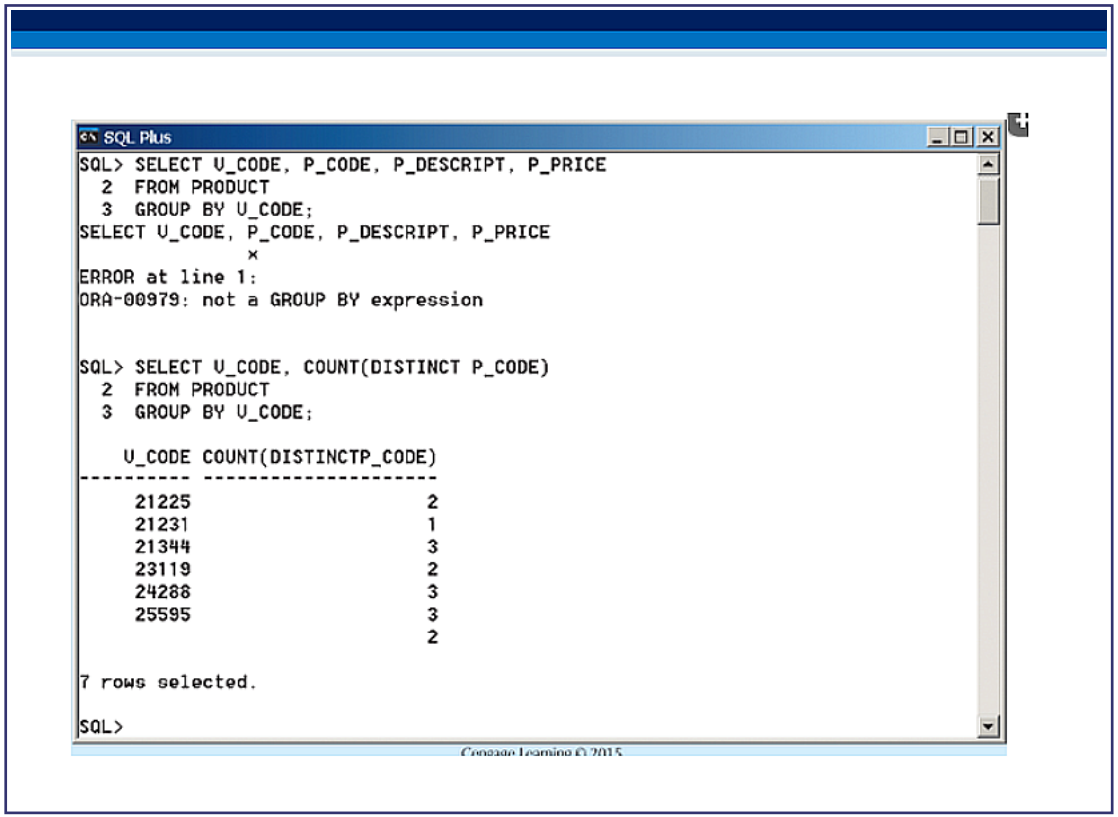
  [HAVING *conditionlist*]

  [ORDER BY *columnlist* [ASC | DESC]];

'GROUP BY' example



You can think of 'GROUP BY' to mean "CATEGORIZE BY" or "ITEMIZE BY": rather than just a single MIN, MAX, SUM, COUNT or AVG, we're asking for a value PER OCCURRENCE of a GROUP BY value, eg. minimum price PER VENDOR, max GPA PER DEPARTMENT, average earnings PER MAJOR, etc. Specifically, when used with SUM(), GROUP BY is used to request subtotals.



'GROUP BY' used without an aggregate function is meaningless, since there is no aggregate value (MIN, COUNT etc.) to itemize.

Store receipt

Here is a grocery store receipt where subtotals are displayed itemized, aggregated by product types DELI, GROCERY, MIXED NUTS and PRODUCE (presumably using SUM(), together with GROUP BY):

SK
SUPER KING
MARKETS
SUPER KING

Join "Royal Rewards" today
and Start Earning Points!

2716 N. San Fernando Rd.
Los Angeles, CA. 90065
(323) 225-0044
Store Hours 7 am - 9 pm
www.SuperKingMarkets.com

Purchase $ 8.74

PIN Used
Debit Card #SXXXXXXXXXXXX4428
Auth # 568771 Payment from primary
Lane # 06 Cashier # 766
01/29/16 09:34 Ref/Seq # 065808
Mrch=912871 Term=001 IC=DC
EPS Sequence # 065808

DELI
SHAMROCK 2% STRAWBER 1.89 *

GROCERY
DORITOS SPICY NACHO
1 @ 2 FOR 5.00 2.50 *

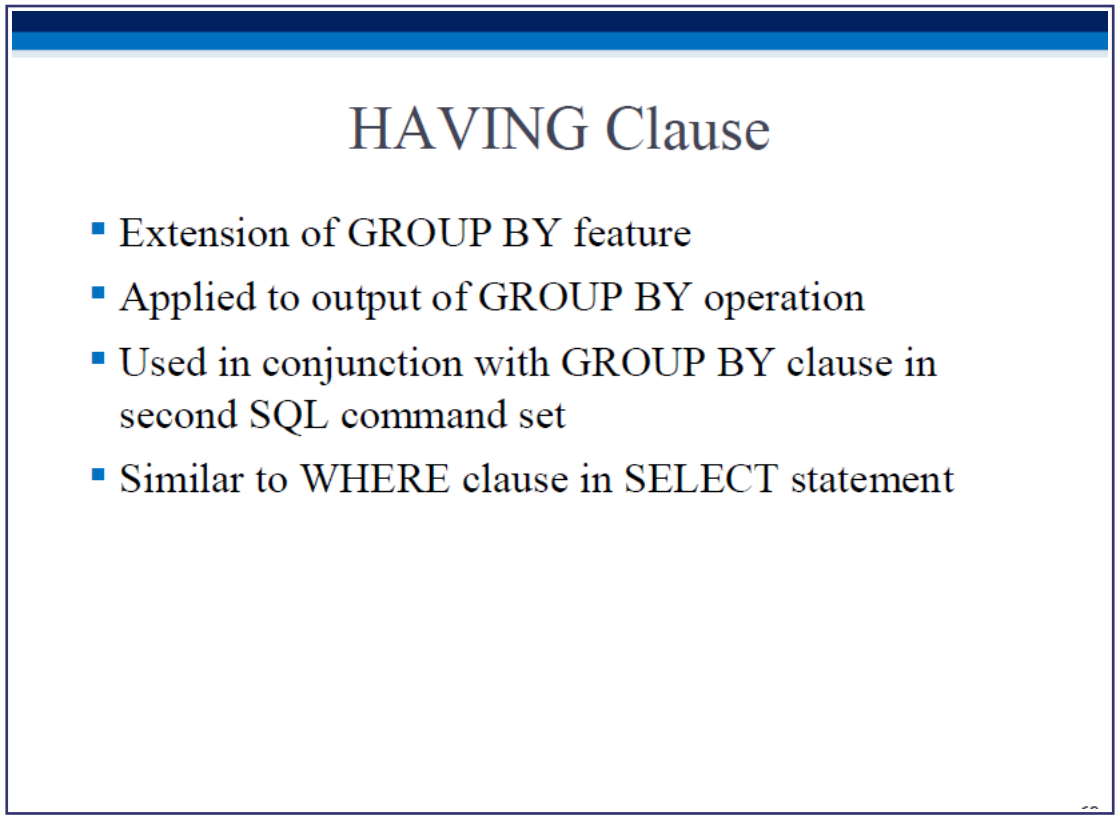
MIXED NUTS
FRIED RED SKIN PEANU
0.66 lb @ 1.99/ lb 1.31 *

While you're at it, see what other DATA you can spot in the receipt! Even a single trip to the store can generate a LOT of data.

HAVING

The HAVING clause is used to filter rows in a GROUP BY specification (only those rows HAVING met the specified condition will appear in the result).

Note that HAVING can only occur in a query that has a GROUP BY, which in turn can only occur when there is an aggregate function (MIN, MAX, SUM, COUNT or AVG).



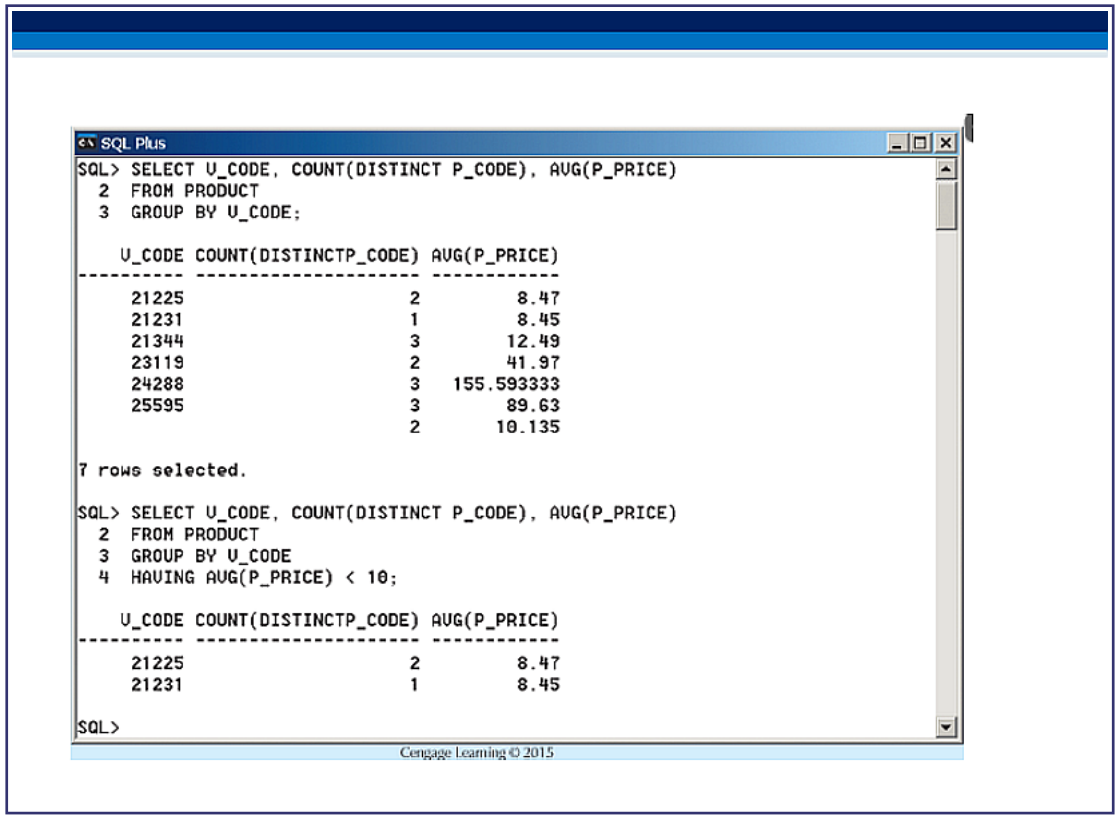

SQL Plus
SQL> SELECT V_CODE, COUNT(DISTINCT P_CODE), AVG(P_PRICE)
2 FROM PRODUCT
3 GROUP BY V_CODE;
V_CODE COUNT(DISTINCTP_CODE) AVG(P_PRICE)
21225 2 8.47
21231 1 8.45
21344 3 12.49
23119 2 41.97
24288 3 155.593333
25595 3 89.63
2 10.135
7 rows selected.
SQL> SELECT V_CODE, COUNT(DISTINCT P_CODE), AVG(P_PRICE)
2 FROM PRODUCT
3 GROUP BY V_CODE
4 HAVING AVG(P_PRICE) < 10;
V_CODE COUNT(DISTINCTP_CODE) AVG(P_PRICE)
21225 2 8.47
21231 1 8.45
SQL>
Cengage Learning © 2015

Table joins

Joining Database Tables

- Performed when data are retrieved from more than one table at a time
  - Equality comparison between foreign key and primary key of related tables
- Tables are joined by listing tables in FROM clause of SELECT statement
  - DBMS creates Cartesian product of every table in the FROM clause

Joining Tables With an Alias

- Alias identifies the source table from which data are taken
- Any legal table name can be used as alias
- Add alias after table name in FROM clause
  - FROM tablename alias, eg. 'FROM PRODUCT P'

```
SELECT    P_DESCRIPT, P_PRICE, V_NAME, V_CONTACT, V_AREACODE, V_PHONE
FROM      PRODUCT, VENDOR
WHERE     PRODUCT.V_CODE = VENDOR.V_CODE;
```

P_DESCRIPT	P_PRICE	V_NAME	V_CONTACT	V_AREACODE	V_PHONE
Claw hammer	9.95	Bryson, Inc.	Smithson	615	223-3234
1.25-in. metal screw, 25	6.99	Bryson, Inc.	Smithson	615	223-3234
2.5-in. wd. screw, 50	8.45	D&E Supply	Singh	615	228-3245
7.25-in. pwr. saw blade	14.99	Gomez Bros.	Ortega	615	889-2546
9.00-in. pwr. saw blade	17.49	Gomez Bros.	Ortega	615	889-2546
Rat-tail file, 1/8-in. fine	4.99	Gomez Bros.	Ortega	615	889-2546
Hrd. cloth, 1/4-in., 2x50	39.95	Randsets Ltd.	Anderson	901	678-3998
Hrd. cloth, 1/2-in., 3x50	43.99	Randsets Ltd.	Anderson	901	678-3998
B&D jigsaw, 12-in. blade	109.92	ORDVA, Inc.	Hakford	615	898-1234
B&D jigsaw, 8-in. blade	99.87	ORDVA, Inc.	Hakford	615	898-1234
Hicut chain saw, 16 in.	256.99	ORDVA, Inc.	Hakford	615	898-1234
Power painter, 15 psi., 3-nozzle	109.99	Rubicon Systems	Orton	904	456-0092
B&D cordless drill, 1/2-in.	38.95	Rubicon Systems	Orton	904	456-0092
Steel matting, 4'x8'x1/6", .5" mesh	119.95	Rubicon Systems	Orton	904	456-0092

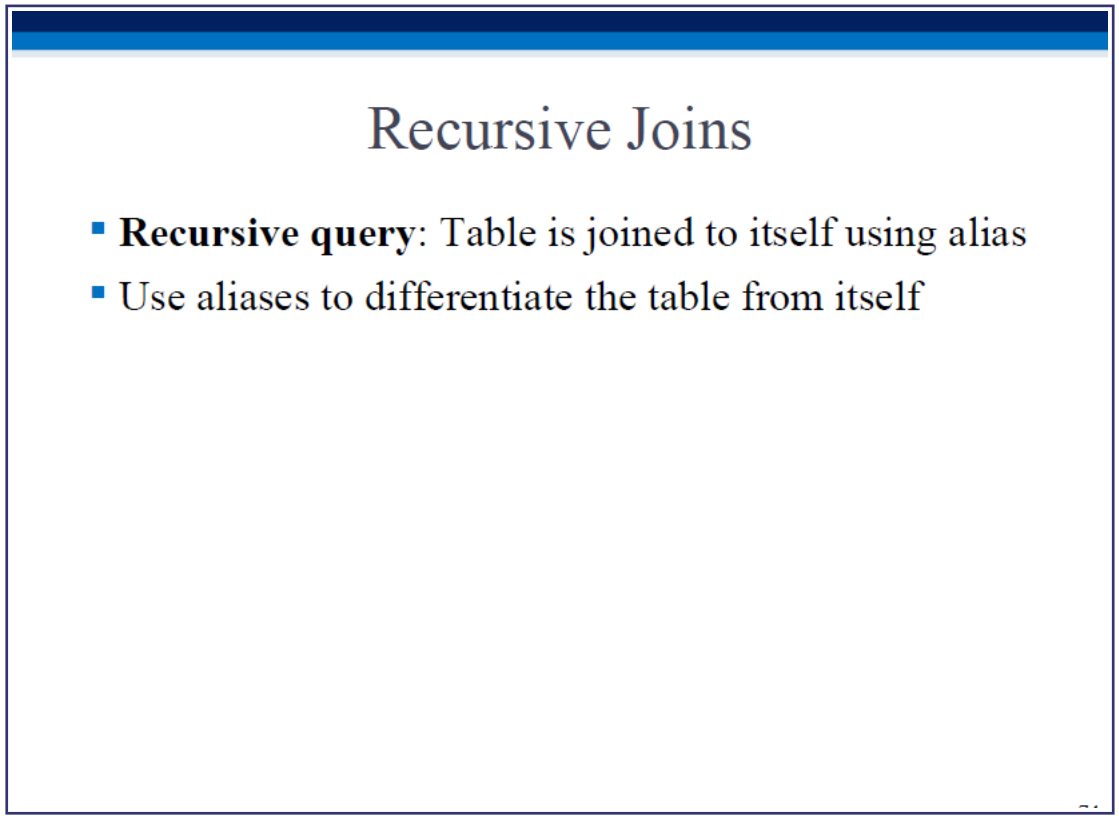

```
ORDER BY    PRODUCT.P_PRICE;
```

```
SELECT      CUS_LNAME, INVOICE.INV_NUMBER, INV_DATE, P_DESCRIPT
FROM        CUSTOMER, INVOICE, LINE, PRODUCT
WHERE       CUSTOMER.CUS_CODE = INVOICE.CUS_CODE
AND         INVOICE.INV_NUMBER = LINE.INV_NUMBER
AND         LINE.P_CODE = PRODUCT.P_CODE
AND         CUSTOMER.CUS_CODE = 10014
ORDER BY    INV_NUMBER;
```

Joining n tables will need (n-1) join conditions.

'Recursive join' example

Need different aliases for the table being queried so that we can use such aliases as namespaces for attributes.



Table, for self-join example

EMP_NUM	EMP_TITLE	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_DOB	EMP_HIRE_DATE	EMP_AREACODE	EMP_PHONE	EMP_MGR
100	Mr.	Kolmycz	George	D	15-Jun-42	15-Mar-85	615	324-5456	
101	Ms.	Lewis	Rhonda	G	19-Mar-65	25-Apr-86	615	324-4472	100
102	Mr.	Vandam	Rhett		14-Nov-58	20-Dec-90	901	675-8993	100
103	Ms.	Jones	Anne	M	16-Oct-74	28-Aug-94	615	898-3456	100
104	Mr.	Lange	John	P	08-Nov-71	20-Oct-94	901	504-4430	105
105	Mr.	Williams	Robert	D	14-Mar-75	08-Nov-98	615	890-3220	
106	Mrs.	Smith	Jeanine	K	12-Feb-68	05-Jan-89	615	324-7883	105
107	Mr.	Diante	Jorge	D	21-Aug-74	02-Jul-94	615	890-4567	105
108	Mr.	Wiesenbach	Paul	R	14-Feb-66	18-Nov-92	615	897-4358	
109	Mr.	Smith	George	K	18-Jun-61	14-Apr-89	901	504-3339	108
110	Mrs.	Genkazi	Leighla	W	19-May-70	01-Dec-90	901	569-0093	108
111	Mr.	Washington	Rupert	E	03-Jan-66	21-Jun-93	615	890-4925	105
112	Mr.	Johnson	Edward	E	14-May-61	01-Dec-83	615	898-4387	100
113	Ms.	Smythe	Melanie	P	15-Sep-70	11-May-99	615	324-9006	105
114	Ms.	Brandon	Marie	G	02-Nov-56	15-Nov-79	901	882-0845	108
115	Mrs.	Saranda	Hermine	R	25-Jul-72	23-Apr-93	615	324-5505	105
116	Mr.	Smith	George	A	08-Nov-65	10-Dec-88	615	890-2984	108

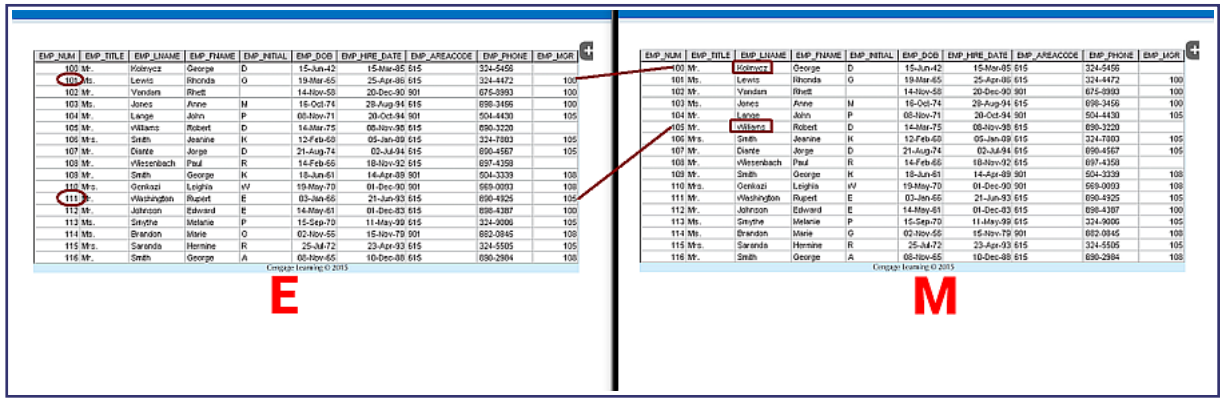
Self-join example

```
SELECT      E.EMP_NUM, E.EMP_LNAME, E.EMP_MGR, M.EMP_LNAME
FROM        EMP E, EMP M
WHERE       E.EMP_MGR=M.EMP_NUM
ORDER BY    E.EMP_MGR;
```

EMP_NUM	E.EMP_LNAME	EMP_MGR	M.EMP_LNAME
112	Johnson	100	Kolmycz
103	Jones	100	Kolmycz
102	Vandam	100	Kolmycz
101	Lewis	100	Kolmycz
115	Saranda	105	Williams
113	Smythe	105	Williams
111	Washington	105	Williams
107	Diante	105	Williams
106	Smith	105	Williams
104	Lange	105	Williams
116	Smith	108	Wiesenbach
114	Brandon	108	Wiesenbach
110	Genkazi	108	Wiesenbach
109	Smith	108	Wiesenbach

Cengage Learning © 2015

Here is how the join works:



Sooo... are joins 'evil'? Not really :)

SQL is declarative, not imperative

Now that you're familiar with SQL syntax, you will appreciate knowing how it compares with a regular programming language such as C++. Shown below is such a comparison, using C# which lets us write code in an imperative (command-oriented) manner as well a declarative (result-oriented) one [this is from a book on functional programming]:

Listing 1.3 Imperative data processing (C#)

```
List<string> res = new List<string>();                          #1
foreach(Product p in Products) {                                #2
   if (p.UnitPrice > 75.0M) {
      res.Add(String.Format("{0} - ${1}",
         p.ProductName, p.UnitPrice));                          #3
   }
}
return res;
```

#1 Create resulting list
#2 Iterate over products
#3 Add information to list of results

You'll probably need to read the code carefully to understand what it does, but that's not the only aspect we want to improve. The code is written as a sequence of some basic imperative commands. For example, the first statement creates new list (#1), the second iterates over all products (#2) and a later one adds element to the list (#3). However, we'd like to be able to describe the problem at a higher level. In more abstract terms, the code just filters a collection and returns some information about every returned product.

In C# 3.0, we can write the same code using query expression syntax. This version is closer to our real goal–it uses the same idea of filtering and transforming the data. You can see the code in listing 1.4.

Listing 1.4 Declarative data processing (C#)

```
var res = from p in Products
          where p.UnitPrice > 75.0M                             #1
          select string.Format("{0} - ${1}",
             p.ProductName, p.UnitPrice);                       #2
return res;
```

#1 Filter products using predicate
#2 Return information about product

The expression that calculates the result (`res`) is composed from basic operators such as `where` or `select`. These operators take other expressions as an argument, because they need to know exactly what we want to filter or select as a result.

Ways to 'do' SQL..

SQL is a data creation+manipulation language, so it's best learned HANDS ON (not just by looking at slides and reading about the syntax) - you need access to a relational database where you can **create tables, enter data in them and do queries on the data** (tables ← data ← queries).

There are three ways to get your hands on a DB: use a browser page [a server runs the DB software, you simply access it from a page]; install a DB locally on your laptop/tablet/phone; use a 'cloud-based' DB [this is similar to, but more powerful than, accessing a DB via a web page].

---

1. Browser-based SQL IDEs

An easy way to practice running SQL queries is to use browser-based interfaces (nothing to download, no login needed) to create/query databases. Here are some sites that provide this form of access:
* SQL Tutorial: http://www.w3schools.com/sql/default.asp
* ideone: http://www.ideone.com
* sqlfiddle: http://sqlfiddle.com/
* Code School's Try SQL: http://campus.codeschool.com/courses/try-sql/contents
* SQLZOO: http://sqlzoo.net/ - has extensive tutorials
* another tutorial: http://www.sqltutorial.org/
* w3resource: http://www.w3resource.com/sql-exercises/ - more tutorials
* Khan Academy: https://www.khanacademy.org/computer-programming/new/sql
* 'Online SQL interpreter': https://sql-js.github.io/sql.js/examples/GUI/ - JavaScript-based! [here is the source: https://cdnjs.com/libraries/sql.js ; more info: https://sql.js.org/#/]
* small, simple, fun examples: https://sql-playground.wizardzines.com/ [to go with this delightful booklet]

Below are examples for three of the above links.

Bring up the w3schools Tryit Editor , enter and run the following pair of sql command sets one at a time (these commands run on a set of pre-installed (by w3schools.com) collection of tables called the Northwind database) :

```
SELECT City FROM Customers
WHERE Country="Germany";

SELECT City, CustomerID FROM Customers
WHERE Country="Germany" AND CustomerID>60;
```

In ideone , select SQLite (for the choice of language), then enter and run this:

```
-- warmup
create table tble(str varchar(20));
insert into tble values ('Hello world!'),('SQL is fun!!');
select * from tble;


-- recipes
CREATE TABLE recipes (
recipe_id INT NOT NULL,
recipe_name VARCHAR(30) NOT NULL,
PRIMARY KEY (recipe_id),
UNIQUE (recipe_name)
);

INSERT INTO recipes
(recipe_id, recipe_name)
VALUES
(1,"Tacos"),
(2,"Tomato Soup"),
(3,"Grilled Cheese");


-- quick check
SELECT recipe_name
FROM recipes;


-- ingredients
CREATE TABLE ingredients (
```

```
ingredient_id INT NOT NULL,
ingredient_name VARCHAR(30) NOT NULL,
ingredient_price INT NOT NULL,
PRIMARY KEY (ingredient_id),
UNIQUE (ingredient_name)
);

INSERT INTO ingredients
(ingredient_id, ingredient_name, ingredient_price)
VALUES
(1, "Beef", 5),
(2, "Lettuce", 1),
(3, "Tomatoes", 2),
(4, "Taco Shell", 2),
(5, "Cheese", 3),
(6, "Milk", 1),
(7, "Bread", 2);


--  recipe_ingredients
CREATE TABLE recipe_ingredients (
recipe_id int NOT NULL,
ingredient_id INT NOT NULL,
amount INT NOT NULL,
PRIMARY KEY (recipe_id,ingredient_id)
);

INSERT INTO recipe_ingredients
(recipe_id, ingredient_id, amount)
```

```
VALUES
(1,1,1),
(1,2,2),
(1,3,2),
(1,4,3),
(1,5,1),
(2,3,2),
(2,6,1),
(3,5,1),
(3,7,2);


SELECT *
FROM recipes
ORDER BY recipe_id;
```

Here is the recipes db table and its queries again, this time in sqlfiddle. After the page loads (with our SQL commands above, filled in!), click 'Build Schema' on the left, then click 'Run SQL' on the right [and wait for a few seconds for the output].

2. Locally installed DBs

You can install Oracle on your machine, or install the light-weight but powerful DB Browser for SQLite, aka SQLite Browser. On Windows, this offers a 'notebook' environment: https://sqlnotebook.com/ [a mix of SQL and non-SQL commands]. It is also OK to use Postgres, or MySQL; or you could use some other DB...

Here is a quickstart guide for SQLite Browser - we create a table called CosineTable, and fill it with rows containing [angle, cos(angle)] pairs. And here is another clip that shows how to create a simple sqlite DB and query it.

The second clip above, also shows that the .db file that is created, contains data in a SQLite-native binary format [which is public and well-documented]. FYI, there are sqlite API calls in multiple languages that let you access the (binary) data using functions/methods that accept SQL commands as regular strings. This is mixed-language programming that lets us accomplish a lot, because we leverage the power of Java, Python etc., as well as that of SQL. Eg. here's how it works in Python:

```
>>> for row in c.execute('SELECT * FROM stocks ORDER BY price'):
        print row # or we could email this, put it up on an LED display, buy/sell the cheap/price
y stocks..

(u'2006-01-05', u'BUY', u'RHAT', 100, 35.14)
(u'2006-03-28', u'BUY', u'IBM', 1000, 45.0)
(u'2006-04-06', u'SELL', u'IBM', 500, 53.0)
(u'2006-04-05', u'BUY', u'MSFT', 1000, 72.0)
```

You could also write a binary reader+parser in C++, Java etc. and extract the sqlite-native data that way - in other words, you'd never lose your data if you store it in a sqlite .db file!

As you can see, sqlite is very popular :)

3. Cloud-based DBs

You can also work with a relational DB via the cloud, eg. using Amazon's RDS offering. RDS offers a way to create databases such as Amazon's own Aurora, Oracle, SQL Server, MariaDB, MySQL, Postgres.

The QUICKEST/EASIEST cloud solution is https://bit.io - give it a try! Here is a good way to get started: https://innerjoin.bit.io/sql-snow-day-a-holiday-adventure-in-postgres-basics-with-bit-io-77f294bb9bed Also, check out https://bit.io/integrations - cool!