Advanced SQL

Case Study Tables [salesperson, product, sales]

- S (<u>SID</u>, SName, City)
- P (<u>PID</u>, PName, Size, Price)
- SP (<u>*SID, PID*</u>, Qty)

Keys ?

Copyright Dan Brandon

Salesperson Table (S)

SID	Sname	City
S1	Peterson	Aarhus
S2	Olsen	Copenhagen
S4	Hansen	Odense
S5	Jensen	Copenhagen

Copyright Dan Brandon

Product Table (P)

PID	PName	Size	Price
P1	Shirt	6	50
P3	Trousers	5	90
P4	Socks	7	20
P5	Blouse	6	50
P8	Blouse	8	60

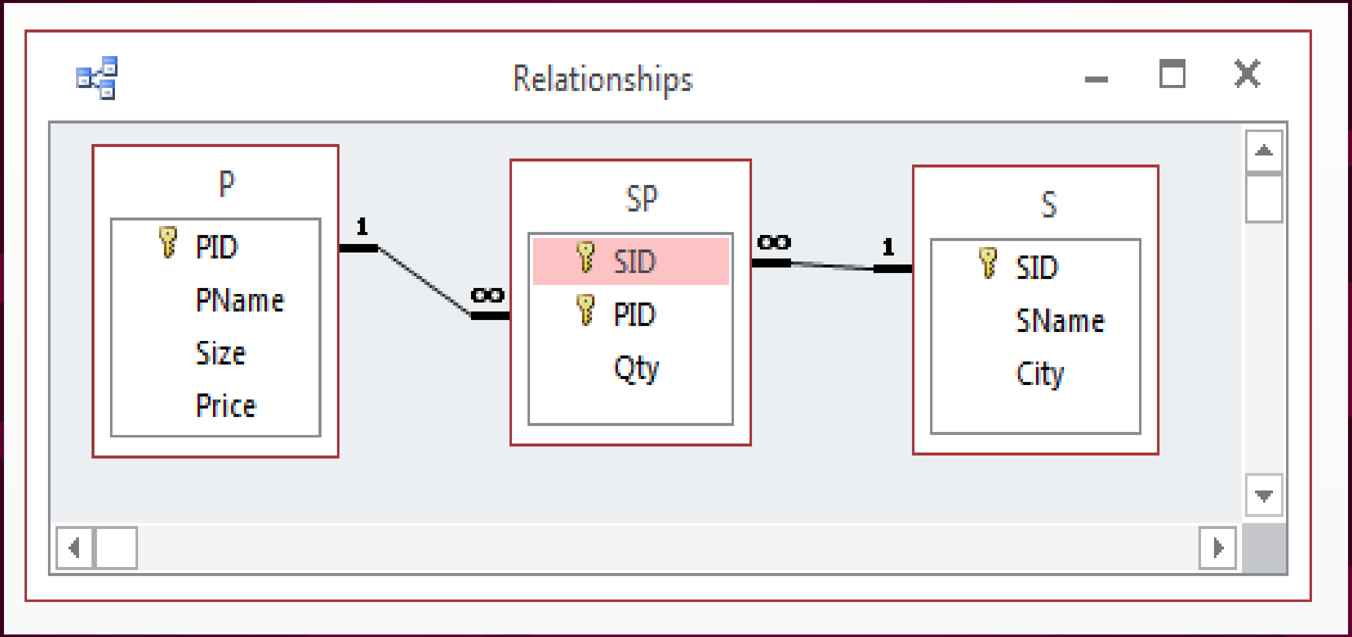
Copyright Dan Brandon

SP Table (Intersection Table)

SID	PID	Qty
S2	P1	200
S2	P3	100
S4	P5	200
S4	P8	100
S5	P1	50
S5	P3	500
S5	P4	800
S5	P5	500
S5	P8	100

Copyright Dan Brandon

Access Relationship Grid



Download this Access file (SP) form the online syllabus.

Copyright Dan Brandon

Function Extensions

- Various extended functions are available in RDBMS's, but the names and syntax varies by product:
  - Date and conversions from/to other types
  - Date/time and conversions from/to other types
  - Numeric functions such as abs, round, floor, ceiling
  - String functions such as concatenation, case changes, substring, length, and conversiosn to/from numeric
  - Sequences
  - Switch
  - Case
  - Window

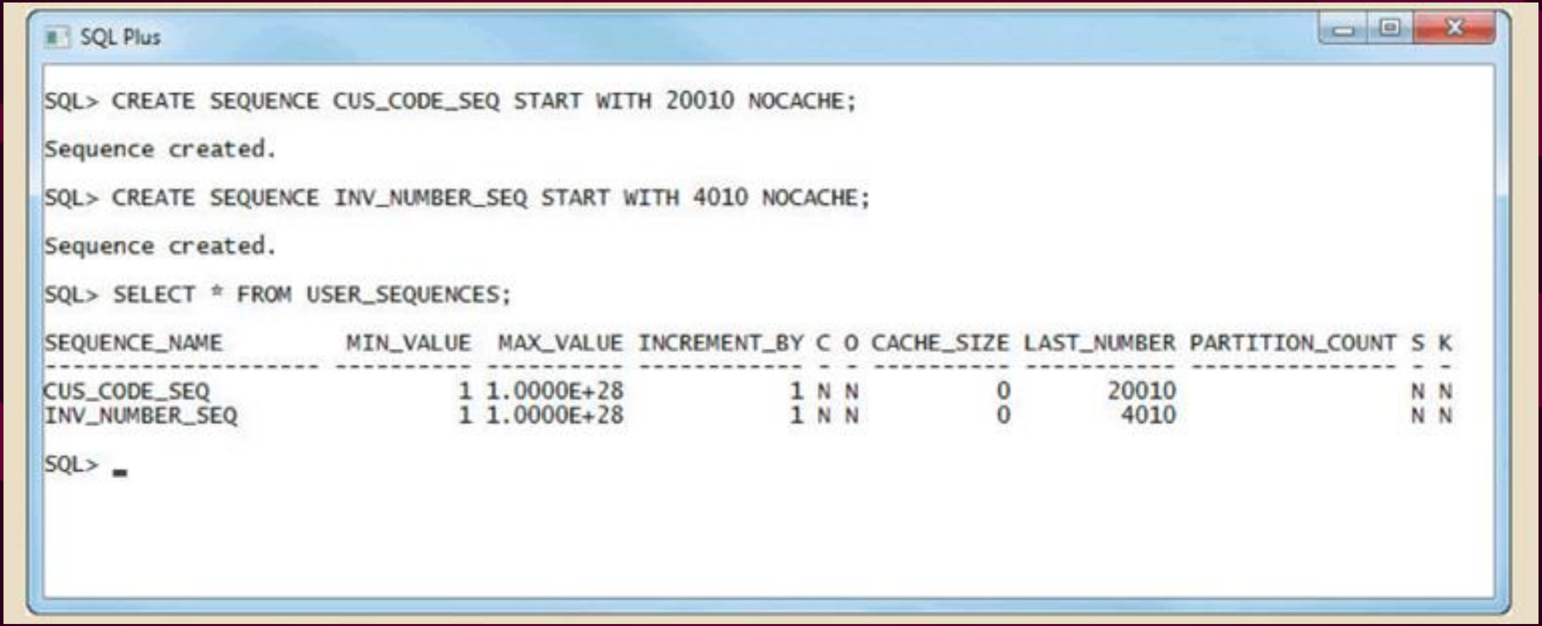
Copyright Dan Brandon

Sequences

- Sequences are an independent object in the database
  - Have a name and can be used anywhere a value expected
  - Not tied to a table or column
  - Generate a numeric value that can be assigned to any column in any table
  - Table attribute with an assigned value can be edited and modified
  - Supported in many RDBMS’s, but not in Access
  - Can be used for auto-increment (identity) column for databases without that built-in capability

Sequences (con't)

- CREATE SEQUENCE seq-name [START WITN n] [INCREMENT BY n] [CACHE | NOCACHE]
  - NOCACHE is 2 words in MS SQL Server
  - INSERT INTO table VALUES (seq-name.NEXTVAL, …)

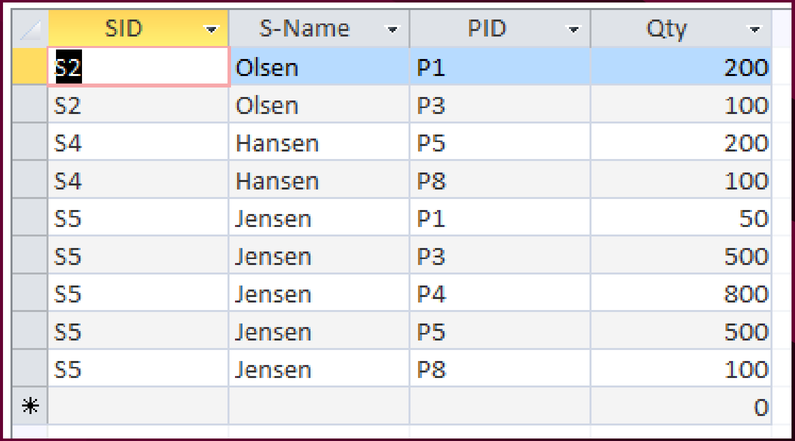


SQL Plus

```
SQL> CREATE SEQUENCE CUS_CODE_SEQ START WITH 20010 NOCACHE;

Sequence created.

SQL> CREATE SEQUENCE INV_NUMBER_SEQ START WITH 4010 NOCACHE;

Sequence created.

SQL> SELECT * FROM USER_SEQUENCES;

SEQUENCE_NAME         MIN_VALUE  MAX_VALUE INCREMENT_BY C O CACHE_SIZE LAST_NUMBER PARTITION_COUNT S K
-------------------- ---------- ---------- ------------ - - ---------- ----------- --------------- - -
CUS_CODE_SEQ                  1 1.0000E+28            1 N N          0       20010                 N N
INV_NUMBER_SEQ                1 1.0000E+28            1 N N          0        4010                 N N

SQL> _
```

Switch

- SELECT SP.SID, Switch([SID]="S1","Peterson",[SID]="S2", "Olsen",[SID]="S4","Hansen",[SID]="S5"," Jensen") AS [S-Name], SP.PID, SP.Qty
- FROM SP;



SID	S-Name	PID	Qty
S2	Olsen	P1	200
S2	Olsen	P3	100
S4	Hansen	P5	200
S4	Hansen	P8	100
S5	Jensen	P1	50
S5	Jensen	P3	500
S5	Jensen	P4	800
S5	Jensen	P5	500
S5	Jensen	P8	100
*			0

Copyright Dan Brandon

CASE…WHEN

- SELECT SID, PID, Qty,
- CASE WHEN Qty>10 THEN ‘Yes’ ELSE NULL END AS ‘Quota Met’
- FROM SP

- CASE is not in Microsoft Access SQL

Copyright Dan Brandon

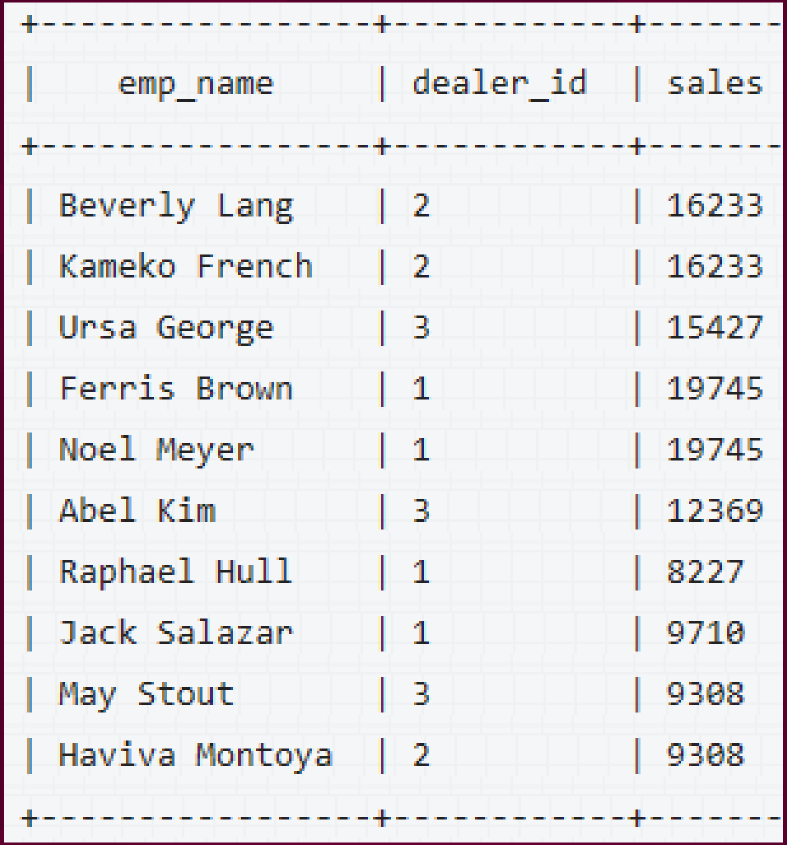
SQL Window Function

- The SQL *window function* performs a calculation across a set of table rows that are somehow related to the current row
- This is comparable to the type of calculation that can be done with an aggregate function such as SUM
- But unlike regular aggregate functions, use of a window function does not cause rows to become grouped into a single output row — the rows retain their separate identities
- Behind the scenes, the window function is able to access more than just the current row of the query result

Window Syntax

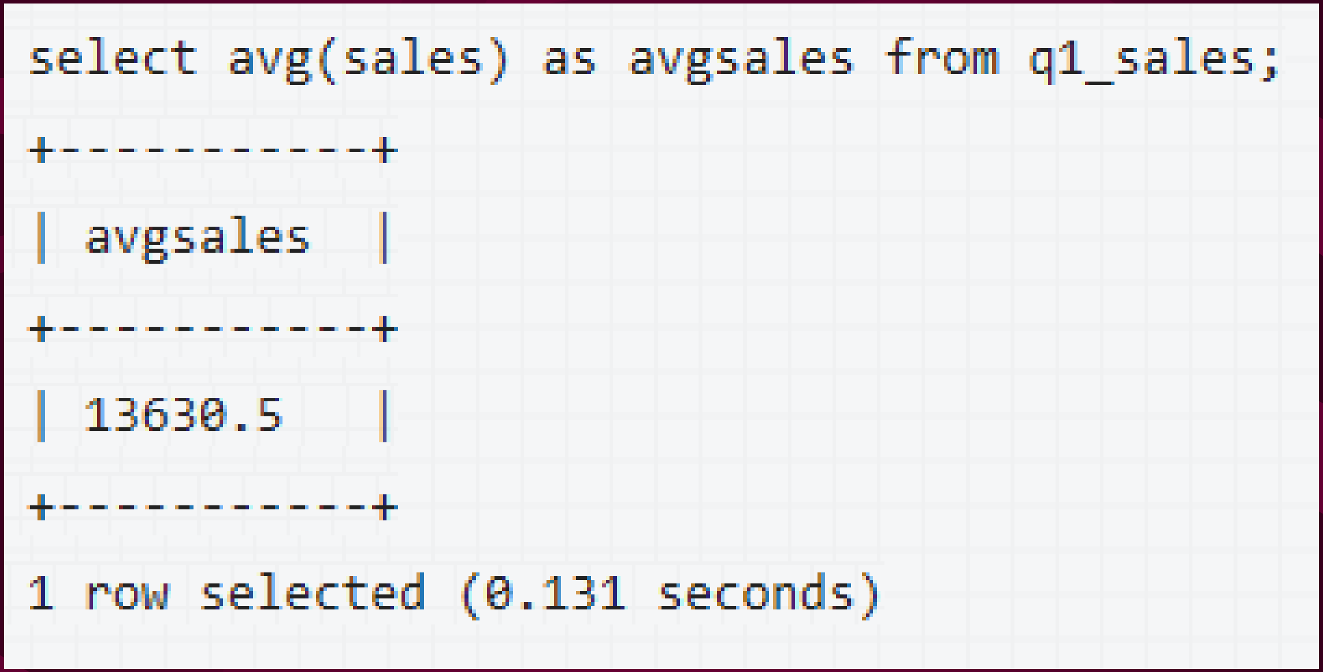
- <window function> OVER
- ([PARTITION BY <expression list>]
- [ORDER BY <expression [ASC|DESC] list>]
- [ROWS|RANGE <window frame>])

Some Sample Data in a Table



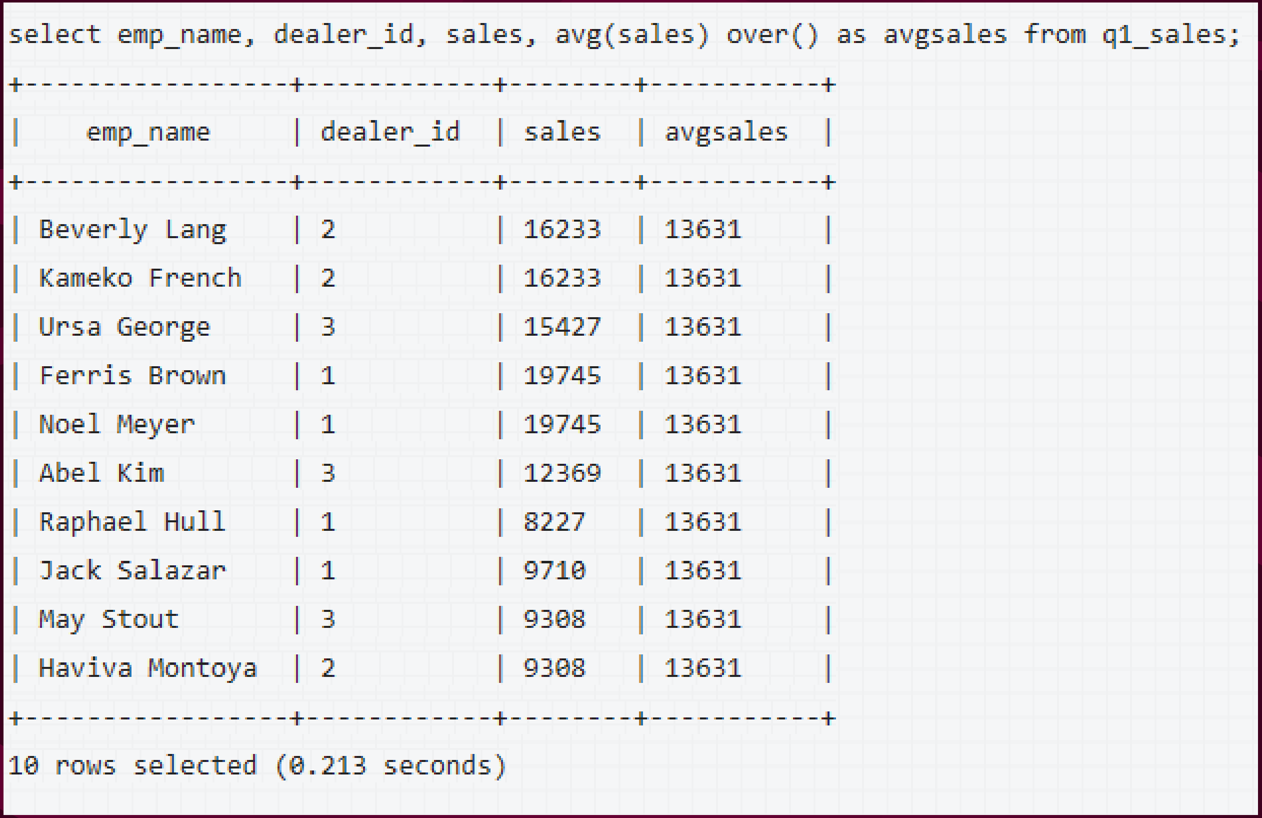
```
+-----------------+------------+-------
|    emp_name     | dealer_id  | sales
+-----------------+------------+-------
| Beverly Lang    | 2          | 16233
| Kameko French   | 2          | 16233
| Ursa George     | 3          | 15427
| Ferris Brown    | 1          | 19745
| Noel Meyer      | 1          | 19745
| Abel Kim        | 3          | 12369
| Raphael Hull    | 1          | 8227
| Jack Salazar    | 1          | 9710
| May Stout       | 3          | 9308
| Haviva Montoya  | 2          | 9308
+-----------------+------------+-------
```

Traditional Aggregate Function (AVG)



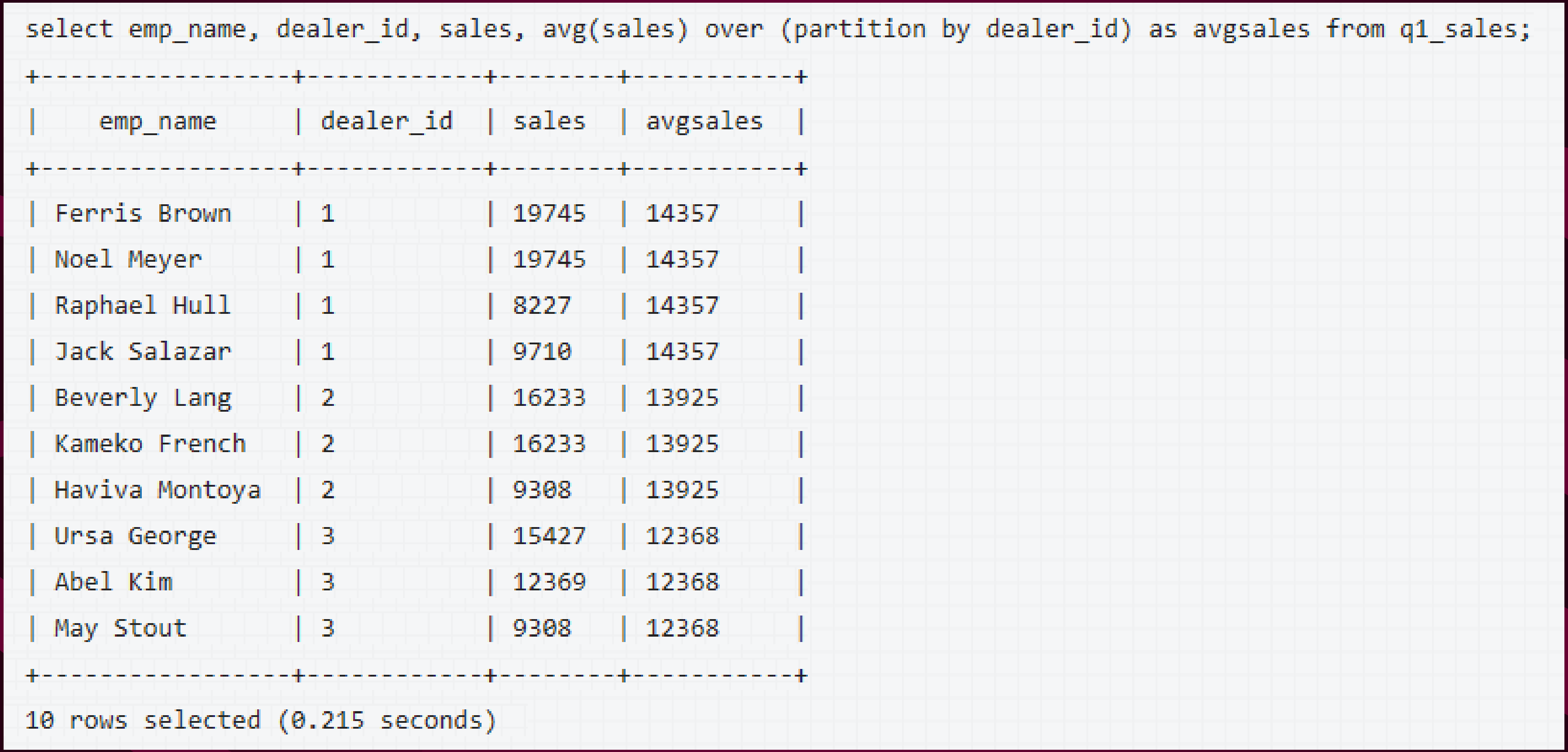
```
select avg(sales) as avgsales from q1_sales;
+------------+
| avgsales   |
+------------+
| 13630.5    |
+------------+
1 row selected (0.131 seconds)
```

Simple Window Query for Running Average



```
select emp_name, dealer_id, sales, avg(sales) over() as avgsales from q1_sales;

+-----------------+------------+--------+-----------+
|    emp_name     | dealer_id  | sales  | avgsales  |
+-----------------+------------+--------+-----------+
| Beverly Lang    | 2          | 16233  | 13631     |
| Kameko French   | 2          | 16233  | 13631     |
| Ursa George     | 3          | 15427  | 13631     |
| Ferris Brown    | 1          | 19745  | 13631     |
| Noel Meyer      | 1          | 19745  | 13631     |
| Abel Kim        | 3          | 12369  | 13631     |
| Raphael Hull    | 1          | 8227   | 13631     |
| Jack Salazar    | 1          | 9710   | 13631     |
| May Stout       | 3          | 9308   | 13631     |
| Haviva Montoya  | 2          | 9308   | 13631     |
+-----------------+------------+--------+-----------+

10 rows selected (0.213 seconds)
```

Partition By Dealer ID



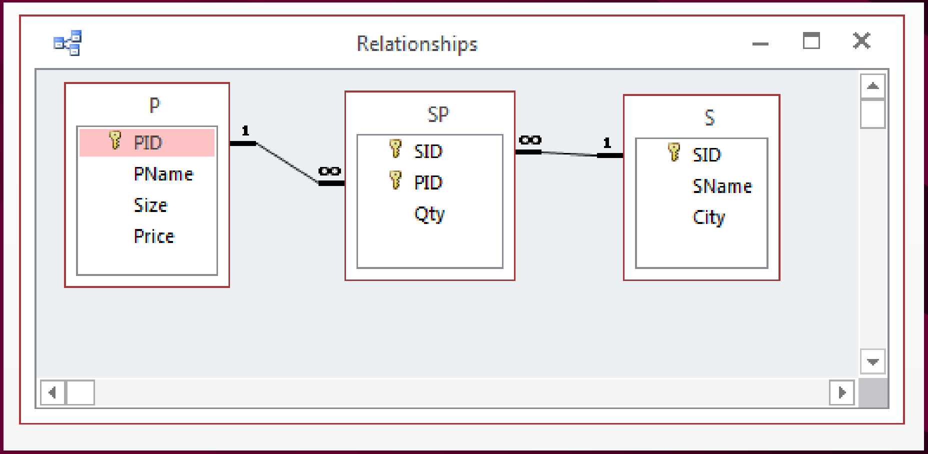
```
select emp_name, dealer_id, sales, avg(sales) over (partition by dealer_id) as avgsales from q1_sales;

+-----------------+------------+--------+-----------+
|    emp_name     | dealer_id  | sales  | avgsales  |
+-----------------+------------+--------+-----------+
| Ferris Brown    | 1          | 19745  | 14357     |
| Noel Meyer      | 1          | 19745  | 14357     |
| Raphael Hull    | 1          | 8227   | 14357     |
| Jack Salazar    | 1          | 9710   | 14357     |
| Beverly Lang    | 2          | 16233  | 13925     |
| Kameko French   | 2          | 16233  | 13925     |
| Haviva Montoya  | 2          | 9308   | 13925     |
| Ursa George     | 3          | 15427  | 12368     |
| Abel Kim        | 3          | 12369  | 12368     |
| May Stout       | 3          | 9308   | 12368     |
+-----------------+------------+--------+-----------+

10 rows selected (0.215 seconds)
```

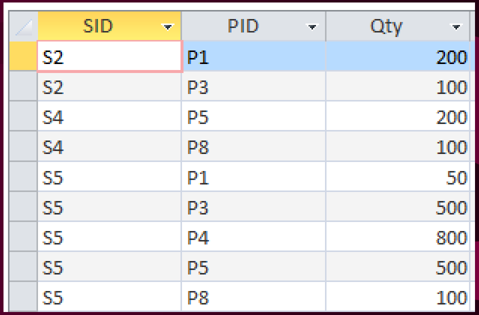
SP Window Example

- SELECT SID, PID, Qty,
- SUM(Qty) OVER (ORDER BY SID) AS Running_total
- FROM SP

"OVER" places a moving "window over the results

SQL Window functions not in Access





SID	PID	Qty
S2	P1	200
S2	P3	100
S4	P5	200
S4	P8	100
S5	P1	50
S5	P3	500
S5	P4	800
S5	P5	500
S5	P8	100

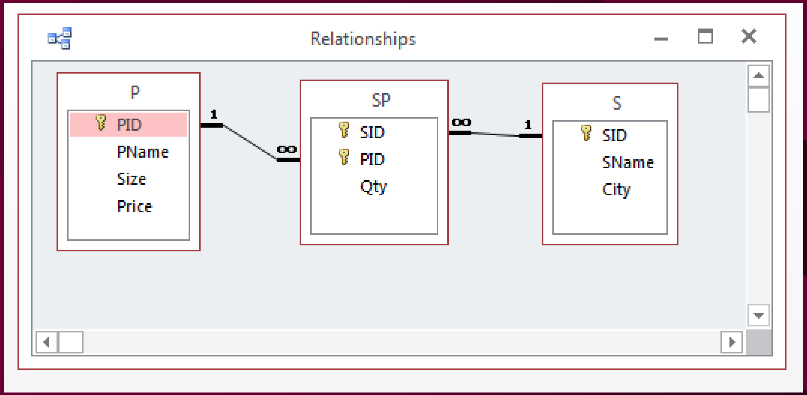
Results…

SID	PID	Running_total
S2	P1	200
S2	P3	300
S4	P5	500
S4	P8	600
S5	P1	650
S5	P3	1150
S5	P4	1950
S5	P5	2450
S5	P8	2550

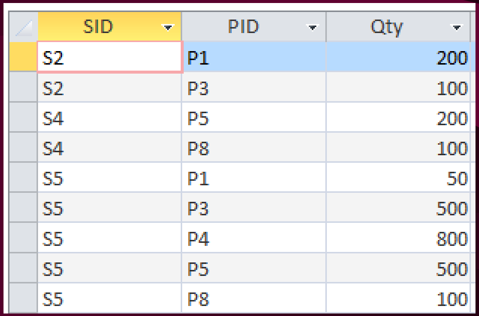
SP Window Example…

- SELECT SID, PID, Qty,
- SUM(Qty) OVER (PARTITION BY SID ORDER BY SID) AS Running_total
- FROM SP

Partition places the moving window just over SID's



Results…



SID	PID	Qty
S2	P1	200
S2	P3	100
S4	P5	200
S4	P8	100
S5	P1	50
S5	P3	500
S5	P4	800
S5	P5	500
S5	P8	100

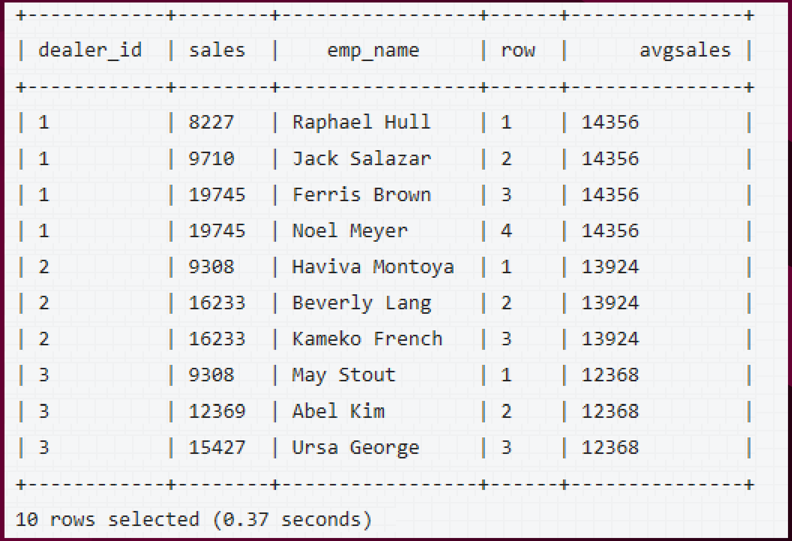
SID	PID	Running_total
S2	P1	200
S2	P3	300
S4	P5	200
S4	P8	300
S5	P1	50
S5	P3	550
S5	P4	1350
S5	P5	1850
S5	P8	1950

Window Notes

- Some database products have can work with a portion of the table/query using ROWS_BETWEEN
- Some database products have an optional *frame_clause*
  - { RANGE | ROWS } frame_start
  - { RANGE | ROWS } BETWEEN frame_start AND frame_end
- Cannot use window functions and standard aggregations (GROUP BY) together
- Common window aggregations are SUM, COUNT, AVG, MIN, MAX
- There is also a ROW_NUMBER() function

Adding Row Number

- select dealer_id, sales, emp_name, row_number() over (partition by dealer_id order by sales) as `row`,avg(sales) over (partition by dealer_id) as avgsales from q1_sales;



```
+------------+--------+-----------------+------+---------------+
| dealer_id  | sales  |    emp_name     | row  |    avgsales   |
+------------+--------+-----------------+------+---------------+
| 1          | 8227   | Raphael Hull    | 1    | 14356         |
| 1          | 9710   | Jack Salazar    | 2    | 14356         |
| 1          | 19745  | Ferris Brown    | 3    | 14356         |
| 1          | 19745  | Noel Meyer      | 4    | 14356         |
| 2          | 9308   | Haviva Montoya  | 1    | 13924         |
| 2          | 16233  | Beverly Lang    | 2    | 13924         |
| 2          | 16233  | Kameko French   | 3    | 13924         |
| 3          | 9308   | May Stout       | 1    | 12368         |
| 3          | 12369  | Abel Kim        | 2    | 12368         |
| 3          | 15427  | Ursa George     | 3    | 12368         |
+------------+--------+-----------------+------+---------------+
10 rows selected (0.37 seconds)
```

Notes (con't)

- There is also a RANK() function, which is like ROW_NUMBER ecept that duplicates are given the same rank (also DENSE_Rank())
- There is also NTILE() for percentile rankings
- LEAD and LAG can be used to compare adjacent ROWS
- If writing several window functions in the same query, may need to alias a window

Updating with SQL

- INSERT
- DELETE
- UPDATE

Copyright Dan Brandon

INSERT

- To add a new salesperson Smith (number S6) who is in Memphis:
- INSERT INTO S
  - VALUES (‘S6’, ‘Smith’, ‘Memphis’)
- If not all columns have values, need to list the columns and order:
  - INSERT INTO S
    - (SID, SName)
    - VALUES (‘S6’, ‘Smith’)

Copyright Dan Brandon

INSERT using query

- CREATE TABLE TEMP
- INSERT INTO TEMP
  - VALUES
    - SELECT SID
    - FROM S
    - WHERE City = ‘Copenhagen’

Copyright Dan Brandon

Access Action Queries

- For lab and project, you do not have to do ‘action queries’ in Access
- If you do, be careful because the queries will change your tables; best to backup your tables before trying them
- Access syntax is somewhat different from standard SQL (refer to an Access book such as the textbook from MIS 153 class)

Copyright Dan Brandon

Access Syntax for ‘Action Query’

- INSERT INTO TEMP
- SELECT SID AS SID
- FROM S
- WHERE City=‘Copenhagen’;

Copyright Dan Brandon

DELETE

- Single row (use primary key or unique index):
  - DELETE
    - FROM S
    - Where SID = ‘S5’
- Multiple rows:
  - DELETE
    - FROM S
    - Where City = ‘Copenhagen’

Copyright Dan Brandon

UPDATE

- Increase the price of one product (P1):
  - UPDATE P
    - SET Price = Price + 5
    - WHERE PID = ‘P1’
- Change the spelling of Copenhagen to Koebenhavn (multiple rows):
  - UPDATE S
    - SET City = ‘Koebenhavn’
    - WHERE City = ‘Copenhagen’

Copyright Dan Brandon

Class Exercise

- Write the SQL to increase the prices by 10% of all products that were <u>sold</u>

Copyright Dan Brandon

Don’t look ahead !

Copyright Dan Brandon, PhD, PMP

Christian Brothers University

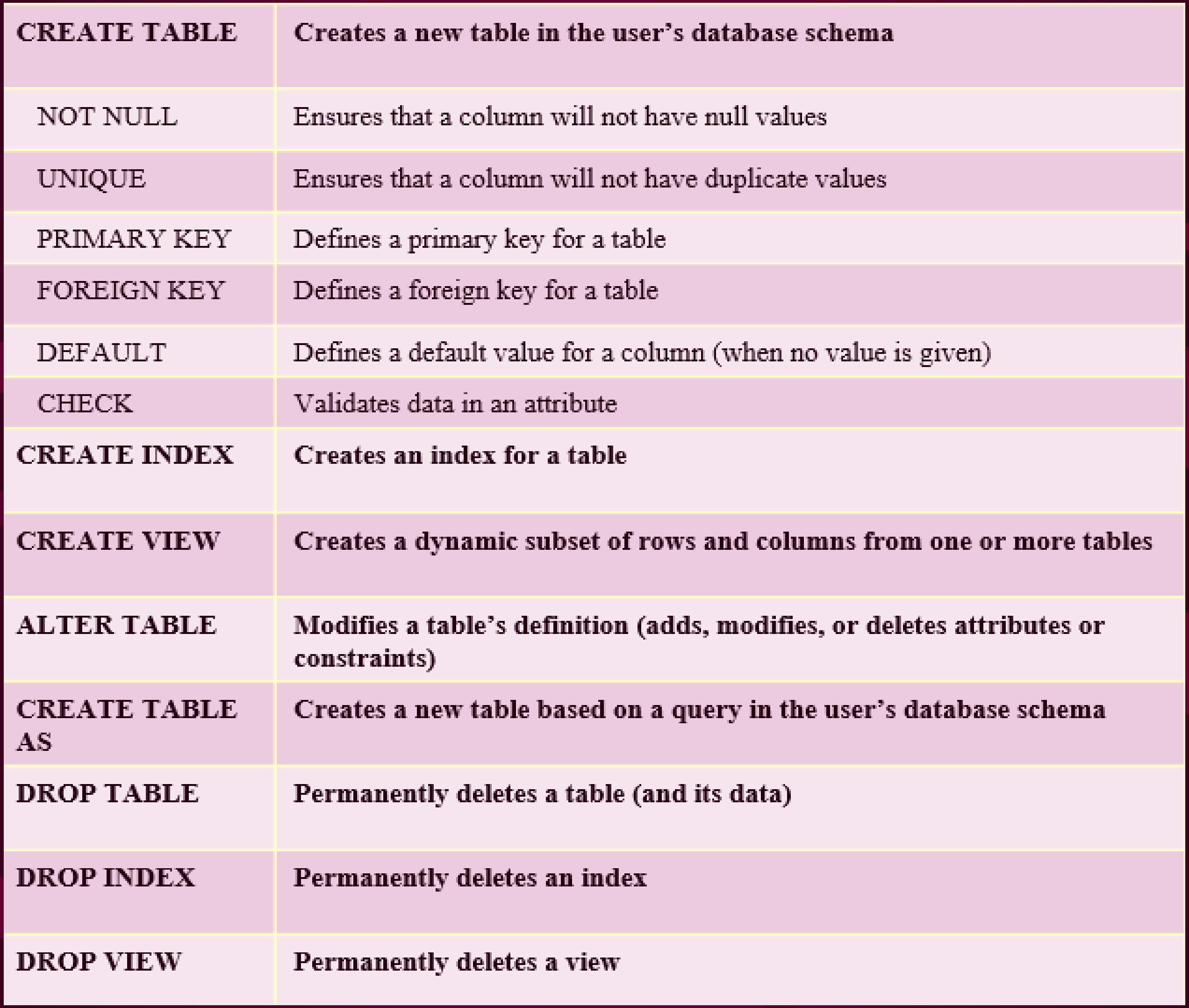
- UPDATE P
  - SET Price = Price * 1.1
  - WHERE EXISTS
    - (SELECT *
      - FROM SP
      - WHERE SP.PID = P.PID)

Stored Procedures

- SQL queries and updates can be "canned" and stored on the server (in a client/server environment)
- Intermediate results can be kept and processed on the server in the same or another stored procedure - reducing network traffic
- Application independence achieved by maintaining common functions one place on the server instead of in each application
- However even though stored procedures use SQL, they also use vendor proprietary scripting languages

Copyright Dan Brandon

SQL Data Definition



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**
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**

SQL Data Definition (con’t)

- CREATE TABLE - Create Table
  - CREATE TABLE
  - CREATE TABLE AS
- DROP TABLE - Delete table and all data
  - DROP TABLE SP
- ALTER TABLE
  - ALTER TABLE S ADD COLUMN STATE CHAR(2) DEFAULT ‘TN’

CREATE TABLE LIKE

- CREATE TABLE XXX LIKE YYY creates an empty table with the same structure as the original table
- CREATE TABLE XXX AS SELECT inserts the data into the new table
  - The resulting table is not empty; only the data is copied not keys and indices
- To create a SQL table from another table without copying any values or keys from the old table:
  - CREATE TABLE XXX
- AS (SELECT * FROM old_table WHERE 1=2);

Relational Model Example

- S (<u>SID</u>, SName, City)
- P (<u>PID</u>, PName, Size, Price)
- SP (<u>*SID, PID*</u>, Qty)
  - or SP(<u>SPID</u>, *SID*, *PID*, Qty)
    - with not null unique index on (SID,PID) and/or (PID,SID)

S (SID, SName, City)

- CREATE TABLE S
  - (SID CHAR(2) NOT NULL,
  - SName VARCHAR (30),
  - City VARCHAR (15),
  - PRIMARY KEY (SID))

P (PID, PName, Size, Price)

- CREATE TABLE P
  - (PID CHAR(2) NOT NULL,
  - PName VARCHAR (20),
  - Size SMALLINT,
  - Price DECIMAL (5.2) NOT NULL,
  - PRIMARY KEY (PID))

SP (*SID, PID*, Qty)

- CREATE TABLE SP
  - (SID CHAR(2) NOT NULL,
  - PID CHAR(2) NOT NULL,
  - Qty INTEGER,
  - PRIMARY KEY (SID, PID),
  - FOREIGN KEY (SID) REFERENCES S,
  - FOREIGN KEY (PID) REFERENCES P)

Table Modifications

- Methods:
  - Alter (simple modifications)
  - Drop and recreate
- Considerations:
  - Existing data
  - Default vales of new data
  - Existing references in programs or queries
- Need for detailed plan !
- Impact minimized by using views for application program and query reference

Typical Table Mod Procedure

- Define new temporary table with changes, new name, and defaults
- Copy data from old table (INSERT INTO X SELECT ...)
- Drop old table (loses views and security also)
- Create new table with indexes, security, etc.
- Copy data from temporary table
- Drop temporary table
- Restore views

DOMAINS

- CREATE DOMAIN CITIES CHAR(15) DEFAULT ‘Memphis’;
- CREATE TABLE S
  - (SID INTEGER NOT NULL,
  - SName VARCHAR (30),
  - City CITIES,
  - PRIMARY KEY (SID))
- Not support by all RDBMS, can keep domains in CASE product

Constraints

- Column
- Table
- Domain
- General

Column Constraints

- NOT NULL
- UNIQUE
  - SSNum CHAR(9) NOT NULL UNIQUE
- DEFAULT
  - State CHAR(2) DEFAULT ‘TN’
- ENUMERATED
  - State CHAR(2) CHECK (State IN (‘TN’, ‘AR’, ‘MS’))
- COMPLEX
  - State CHAR(2) CHECK (State IN SELECT *
  - FROM STATES)

Table Constraints

[multiple columns within the same table]

- Uniqueness (multiple columns)
  - CREATE TABLE PROJECT
    - (ProjNo INTEGER NOT NULL,
    - PID INTEGER NOT NULL,
    - EID INTEGER NOT NULL,
    - PRIMARY KEY (ProjNo)
    - UNIQUE (PID, EID))
- Primary Keys are automatically unique
- Complex (multiple columns)
  - Blouses cannot cost more than $60
  - CONSTRAINT IC01 CHECK
    - (P.Name NOT = ‘Blouse’ or P.Price NOT > 60)

Domain Constraints

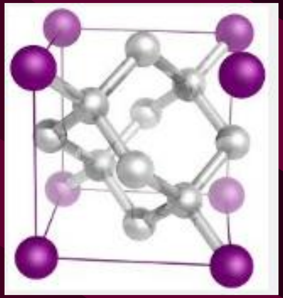
- CREATE DOMAIN CITIES CHAR (15)
  - CHECK (VALUE IS NOT NULL);
- CREATE DOMAIN CITIES CHAR (15)
  - CONSTRAINT MIDSOUTH
  - CHECK (VALUE IN (‘Memphis’, ‘Nashville’, ‘Little Rock’, ‘Jackson’))

General Constraints - ASSERTIONS

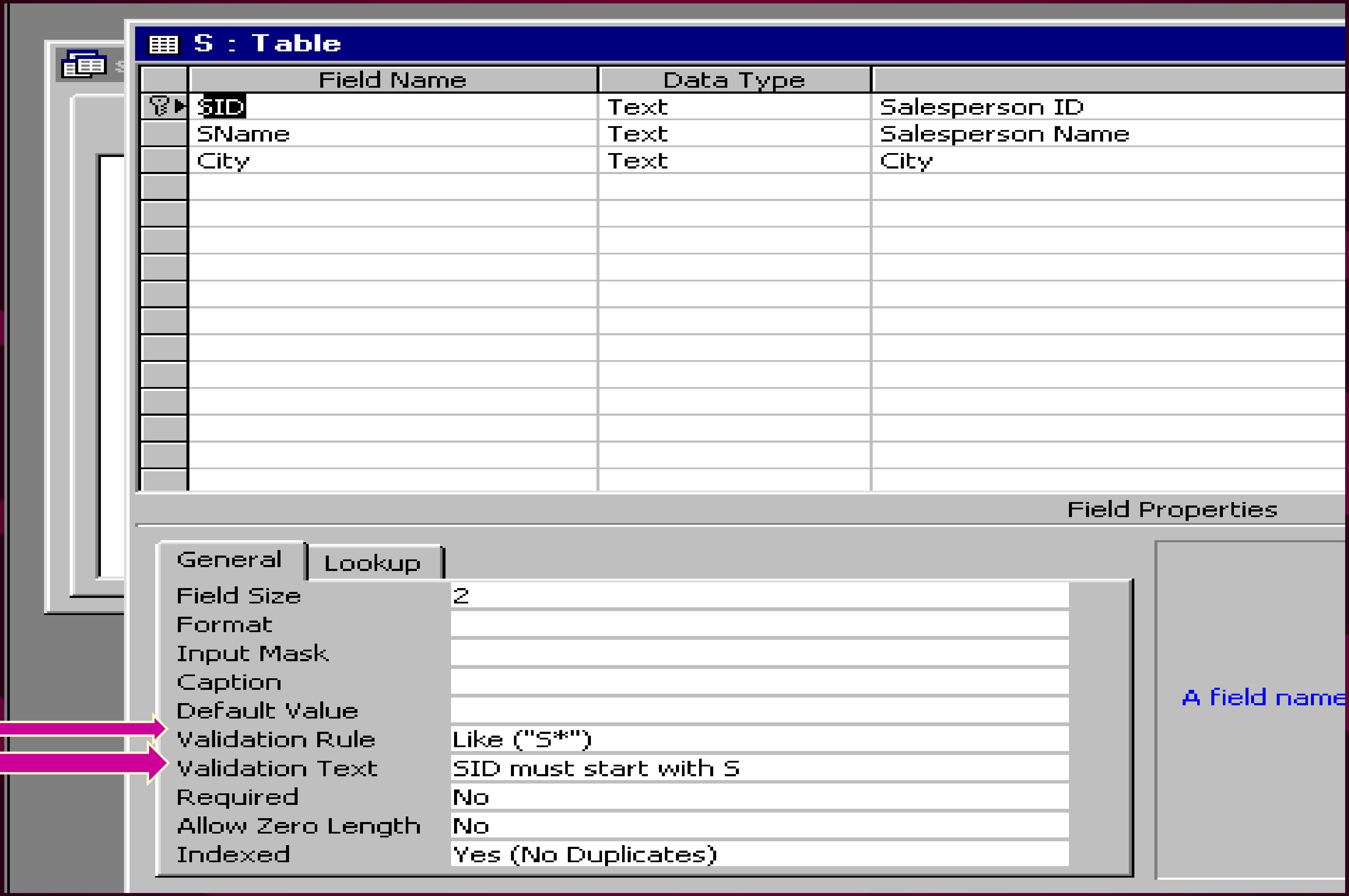
- Any arbitrary combination of tables and columns involved !
- CREATE ASSERTION IC99 CHECK
  - (NOT EXISTS (SELECT * FROM P, SP WHERE P.PID = SP.PID AND (P.Price * SP.Qty) > 100000))
- No salesperson can sell more than 100000 dollars of a product

Constraint Deferral

- Constraints may be initially activated (IMMEDIATE) or deferred
- Later that status may be changed
- Constraints can also be qualified as not deferrable
- Sometimes need for:
  - roots of tree (or lattices)
  - initial data loading for required minimum cardinality



Access Constraints



S : Table

Field Name	Data Type	
SID	Text	Salesperson ID
SName	Text	Salesperson Name
City	Text	City

Field Properties

Field Size	2
Format	
Input Mask	
Caption	
Default Value	
Validation Rule	Like ("S*")
Validation Text	SID must start with S
Required	No
Allow Zero Length	No
Indexed	Yes (No Duplicates)

ValidationRule	**ValidationText**
<> 0	Entry must be a nonzero value
> 1000 Or Is Null	Entry must be blank or greater than
	1000
Like "A????"	Entry must be 5 characters and begin
	with the letter "A"
>= #1/1/96# And <#1/1/97#	Entry must be a date in 1996

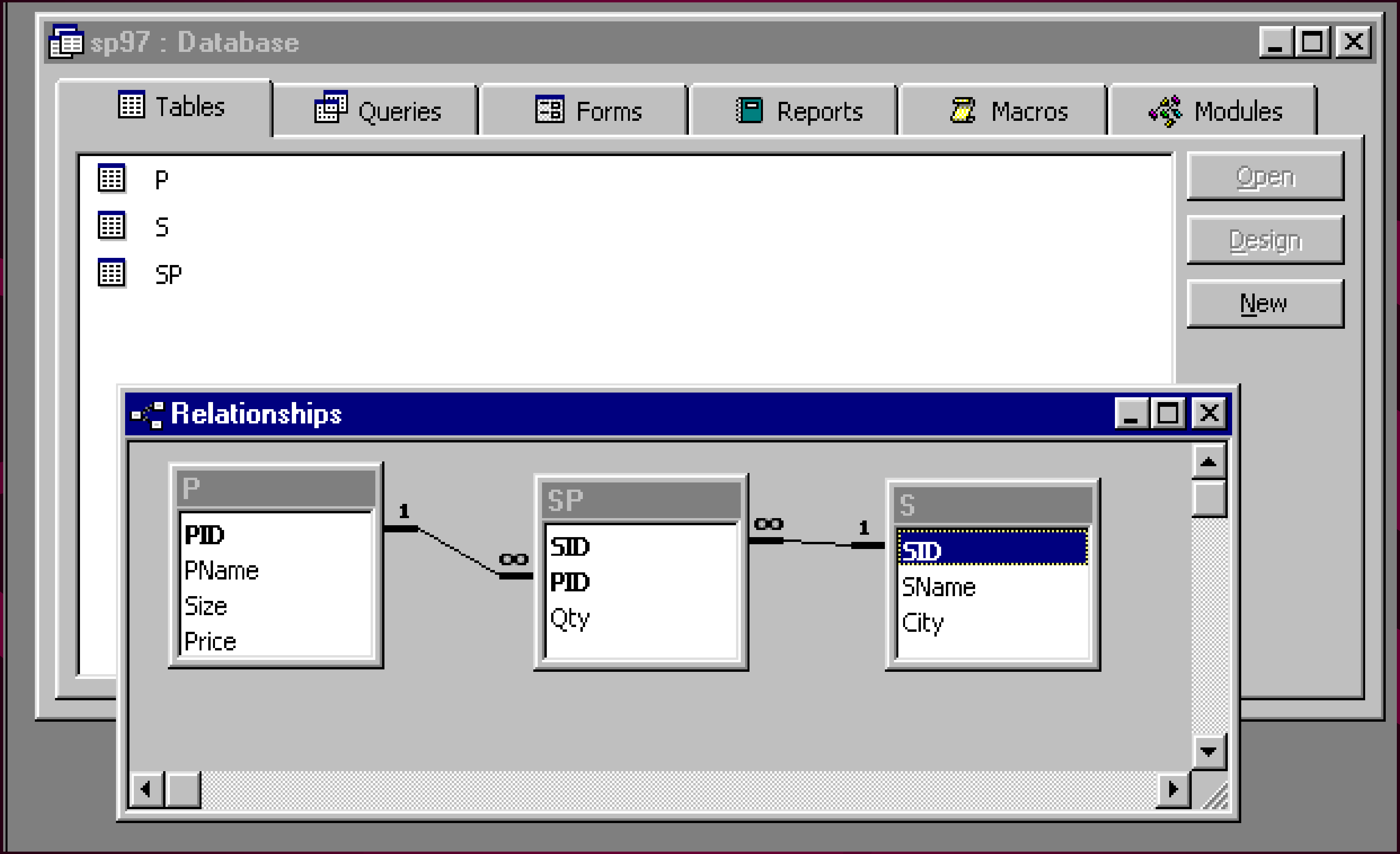
- If you create a validation rule for a field, Microsoft Access doesn't normally allow a Null value to be stored in the field
- If you want to allow a Null value, add "Is Null" to the validation rule, as in "<> 8 Or Is Null" and make sure the Required property is set to No

Referential Integrity Constraints

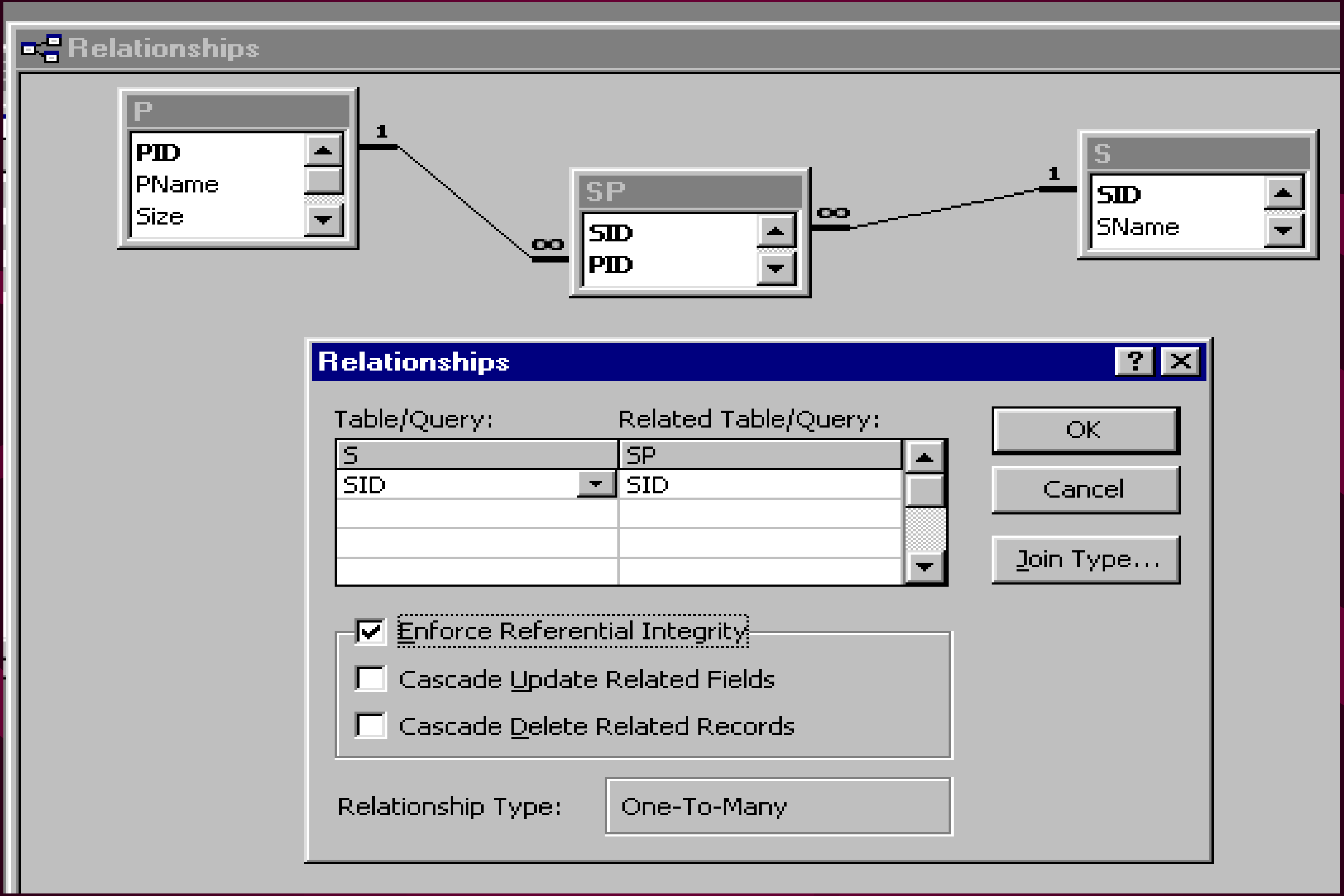
- One may define conditions for deletion and updating of parent entities:
  - Restrict
  - Cascade
  - Set Null
  - Set Default (matching PK must exists)

- CREATE TABLE SP
  - (SID CHAR(2),
  - PID CHAR(2) NOT NULL,
  - Qty INTEGER,
  - PRIMARY KEY (SID, PID),
  - FOREIGN KEY (SID) REFERENCES S ON DELETE SET NULL,
  - {if a salesperson is deleted, his orders have a null salesperson assigned to them}
  - FOREIGN KEY (PID) REFERENCES P ON DELETE CASCADE)
  - {if a product is deleted, all orders for that product are also deleted}

Access Relationship Grid



Referential Integrity in Access



Table/Query:	Related Table/Query:
S	SP
SID	SID

[✓] Enforce Referential Integrity

[ ] Cascade Update Related Fields

[ ] Cascade Delete Related Records

Relationship Type: One-To-Many

Access Cascading Updates and Deletes

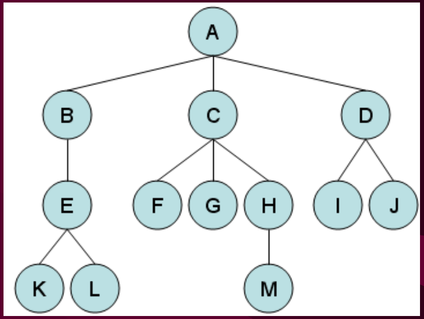
- For relationships in which referential integrity (referential integrity: Rules that you follow to preserve the defined relationships between tables when you enter or delete records.) is enforced, you can specify whether you want Microsoft Access to automatically cascade update (cascading update: For relationships that enforce referential integrity between tables, the updating of all related records in the related table or tables when a record in the primary table is changed.) and cascade delete (cascading delete: For relationships that enforce referential integrity between tables, the deletion of all related records in the related table or tables when a record in the primary table is deleted.) related records. If you set these options, delete and update operations that would normally be prevented by referential integrity rules are allowed. When you delete records or change primary key (primary key: One or more fields (columns) whose values uniquely identify each record in a table. A primary key cannot allow **Null** values and must always have a unique index. A primary key is used to relate a table to foreign keys in other tables.) values in a primary table (primary table: The "one" side of two related tables in a one-to-many relationship. A primary table should have a primary key and each record should be unique.), Microsoft Access makes necessary changes to related tables to preserve referential integrity.
- If you select the **Cascade Update Related Fields** check box when defining a relationship, any time you change the primary key of a record in the primary table, Microsoft Access automatically updates the primary key to the new value in all related records. For example, if you change a customer's ID in the Customers table, the CustomerID field in the Orders table is automatically updated for every one of that customer's orders so that the relationship isn't broken. Microsoft Access cascades updates without displaying any message.
- **Note** If the primary key in the primary table is an AutoNumber (AutoNumber data type: In a Microsoft Access database, a field data type that automatically stores a unique number for each record as it's added to a table. Three kinds of numbers can be generated: sequential, random, and Replication ID.) field, setting the **Cascade Update Related Fields** check box will have no effect, because you can't change the value in an AutoNumber field.
- If you select the **Cascade Delete Related Records** check box when defining a relationship, any time you delete records in the primary table, Microsoft Access automatically deletes related records in the related table. For example, if you delete a customer record from the Customers table, all the customer's orders are automatically deleted from the Orders table (this includes records in the Order Details table related to the Orders records). When you delete records from a form or datasheet with the **Cascade Delete Related Records** check box selected, Microsoft Access warns you that related records may also be deleted. However, when you delete records using a delete query (delete query: A query (SQL statement) that removes rows matching the criteria that you specify from one or more tables.), Microsoft Access automatically deletes the records in related tables without displaying a warning.

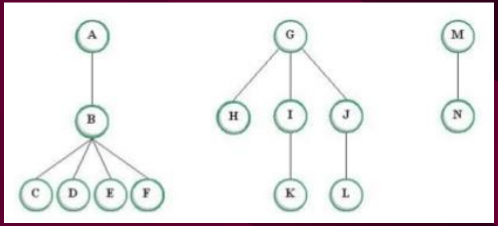
Constraint Application

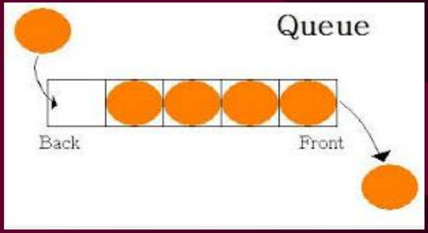
- It may be better to place your constraints in the database rather than in the application program:
  - easier maintenance (one place instead of in every program)
  - enforced for direct SQL usage as well as via application programs
  - more efficient
- However not all database systems handle all referential integrity options, and in an application program you can code whatever type of referential integrity you need and give the user better error messages

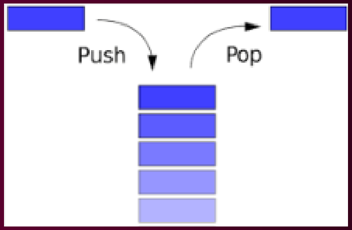
Data Structures and Constraints

- Tree data structure
  - All entries except the top (root) have a master record
- Forest data structure
  - Multiple trees
- Queue
  - FIFO
- Stack
  - LIFO









Referential Integrity Checking in Application Code (Insertion)

```
•  if ($master != "") {  // referential integrity check in PHP/MySql for Web App
•          if ($level == 0) {
•               echo '<h2>Error: If a Master WBS Code is specified,’;
•               echo ‘ the level must not be zero !</h2>';
•               exit;
•          }
•          $query2 = "select * from $tableName where wbsCode='".$master."';";
•          $result2 = mysql_query($query2);
•          $num_results2 = mysql_num_rows($result2);
•          if ($num_results2 == 0) {
                     echo '<h2>Error: Invalid/Non-existent Master WBS Code !</h2>';
•                    exit;
•          }
•  }
```

Class Example

- Given the relations:
  - FACULTY (<u>Fname</u>, Status)
  - ADVISEE (<u>SID</u>, SName, Major, ...)
  - GRAD-ADVISEE (*<u>SID</u>*, *GFname*) ***SubTypes***
  - UGRAD-ADVISEE (*<u>SID</u>*, *FName*)
- Develop SQL constraints for:
  - Status is 0 (undergrad faculty) or 1 (grad faculty)
  - SID for graduate students start with 9, undergrads do not
  - <u>Graduate students only have graduate faculty for advisors</u>

Don’t look ahead !

Copyright Dan Brandon, PhD, PMP

Christian Brothers University

- CREATE TABLE FACULTY
  - …
  - Status   CHAR (1) NOT NULL CHECK (Status IN (0,1))

- CREATE TABLE UGRAD-ADVISEE
  - …
  - SID CHAR (…) NOT NULLCHECK (SID NOT LIKE 9%)

- CREATE TABLE GRAD-ADVISEE
  - SID CHAR (...) NOT NULL CHECK (SID LIKE 9%)
  - GFName CHAR (...) CHECK (GFName IN (SELECT FName FROM FACULTY WHERE Status = 1))

Views [Access Queries]

- A virtual table derived from one or more other tables (or other views)
- Defined via SQL
- <u>View definition is stored but the user data is only stored in the underlying tables</u>
- Use view names in SQL queries just like table names (almost)
- On some systems, some views are updateable (such as those created from one table)

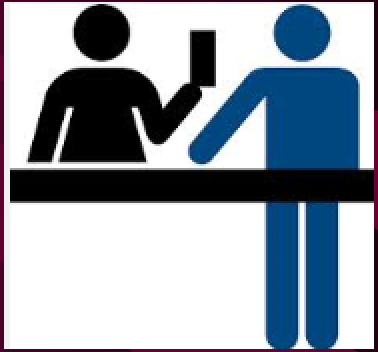
View Usage

- Make queries easy for end users
- Application Independence - applications coded to views instead of actual tables
- Security - certain uses have access to only parts of table(s)

View Syntax

- CREATE VIEW SALESPERSON AS
  - SELECT SName, City
  - FROM S
  - ORDER BY SName
- CREATE VIEW SALES AS
  - SELECT SName, PName, Qty
  - FROM S, SP, P
  - WHERE S.SID = SP.SID and P.PID = SP.PID
  - ORDER BY Pname

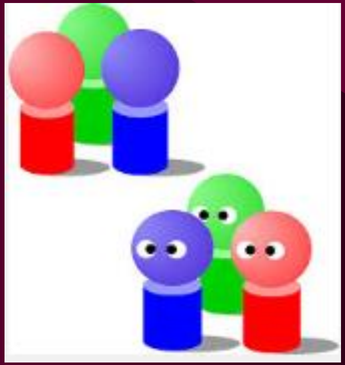
Easy Queries for End Users



- SELECT *
- FROM SALES
- WHERE Qty > 100

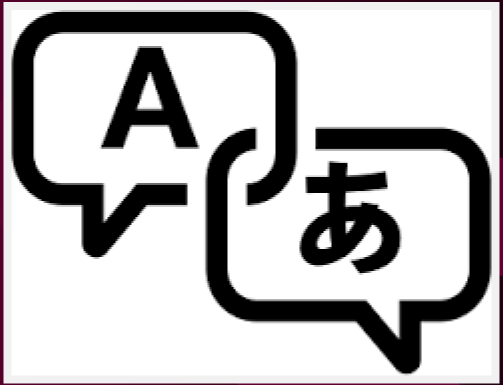
Views with Groups

- CREATE VIEW SALESCOMMISSION (Salesperson, Commission) AS
  - SELECT SName, SUM (Price *Qty *.1)
  - FROM S, SP, P
  - WHERE S.SID = SP.SID and P.PID = SP.PID
  - GROUP BY SID
  - ORDER BY SName



View Translation

- CREATE VIEW EXPENSIVE AS
  - SELECT PName, Price
  - FROM P
  - WHERE Price > 50
- UPDATE EXPENSIVE
  - SET Price = Price + 5
  - WHERE PName = ‘Blouse’
- UPDATE P
  - SET Price = Price + 5
  - WHERE PName = ‘Blouse’
  - AND Price > 50



<u>TRANSLATION</u>

Illegal View Retrievals

- CREATE VIEW PQ (PID, TotQty) AS
  - SELECT PID, SUM (Qty)
  - FROM SP
  - GROUP BY PID
- SELECT PID
  - FROM PQ
  - WHERE TotQty > 500
- SELECT PID *Translation - illegal, need HAVING*
  - FROM SP
  - WHERE SUM (Qty) > 500
  - GROUP BY PID

Illegal View Usage

- A FROM clause that references a grouped view is not allowed to have an associated WHERE clause, GROUP BY clause, or HAVING clause
- SELECT PID *Legal*
  - FROM SP
  - GROUP BY PID
  - HAVING SUM (Qty) > 500

Updateable Views

- It does not include the word distinct
- Every item in the select clause consists of a simple reference to a column of the underlying table (ie it is not a constant, nor an operational expression, nor an expression involving a function)
- The FROM clause identifies exactly one table and that table is updateable
- The WHERE clause does not include a subquery
- There is no GROUP BY or HAVING Clause

Procedural SQL

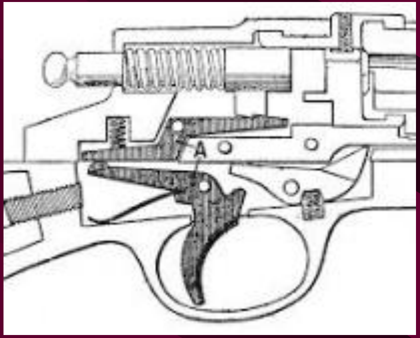
- Performs a conditional or looping operation by isolating critical code and making all application programs call the shared code
  - Better maintenance and logic control
- Persistent stored module (PSM): block of code
  - Contains standard SQL statements and procedural extensions that is stored and executed at the DBMS server
- Use and syntax of procedural SQL varies by vendor

Procedural SQL (con’t)

- Procedural Language SQL (PL/SQL)
  - Use and storage of procedural code and SQL statements within the database
  - Merging of SQL and traditional programming constructs
- Procedural code is executed as a unit by DBMS when invoked by end user for:
  - Triggers
  - Stored procedures
  - PL/SQL functions

Triggers

- A trigger causes an SQL statement (or stored procedure) to be automatically invoked whenever certain events (insert, update, or delete) happen to a table (or a view)
- CREATE TRIGGER T01
  - AFTER UPDATE OF P.OnHand ON P
  - WHEN (P.OnHand < P.ReorderPoint)
  - BEGIN
    - PERFORM Reorder (PID)
  - END



- Also can specify if action is to be performed just once per table or for each row, and maybe able to specify “before”, “after”, or “instead of”
- Triggers are typically not pre compiled, but cached

Trigger Uses

- Synchronization of denormalized tables
- Sums and other computed values of entire tables
- Automatically invoking application level functions; for example, reorder a part whenever the stock on hand falls below the reorder point
- Enforcing business rules that cannot be directly handled with constraints; or checking constraints upon certain database conditions
- PENDANTS - clean up; for example deleting the parent when the last child is deleted

Trigger Cautions

- Users unaware of triggers
  - Example: order clerks using dummy orders to reserve stock
- Trigger/lock conflicts
- Cascading of triggers & infinite loops

Access Triggers (Events)

- The <u>Activate event</u> occurs when a form or report receives the focus and becomes the active window
- The <u>Deactivate event</u> occurs when a form or report loses the focus to a Table, Query, Form, Report, Macro, or Module window, or to the Database window
- Note: The Deactivate event doesn't occur when a form or report loses the focus to a dialog box, to a form for which the PopUp property is set to Yes, or to a window in another application

- To run a macro or event procedure when these events occur, set the OnActivate, or OnDeactivate property to the name of the macro or to [Event Procedure]
- You can make a form or report active by opening it, clicking it or a control on it, or by using the SetFocus method in Visual Basic (for forms only)
- The Activate event can occur only when a form or report is visible
- The Activate event occurs before the GotFocus event; the Deactivate event occurs after the LostFocus event
- Can also have macros triggered from GotFocus, LostFocus, and other events

Stored Procedures

- Named collection of procedural and SQL statements
  - Stored in the database
  - Can be used to encapsulate and represent business transactions
- Advantages
  - Reduce network traffic and increase performance
  - Decrease code duplication by means of code isolation and code sharing
- They are typically written in the proprietary language of the database (but some new versions of products may also allow them to be written in Java)
- They allow functional logic to be maintained within the database instead of the application
- Table changes require recompilation of associated stored procedures

Embedding SQL in Program Code

- SQL works in “sets”, but programs (Java, C++, PHP, etc.) use data in variables, arrays, vectors (or other data structures), or files
- To convert from sets to program variables:
  - Older systems use the concept of “result files” and “cursors”
  - Newer systems represent results sets as arrays of arrays (2 dimensional arrays or tables) or vectors of vectors

SQL in Program Code (Java)

```
public void execSQLCommand(String command)  // the  string argument is an SQL statement
        {
                try     {
                    theStatement = theConnection.createStatement();
                    theResult = theStatement.executeQuery(command); // vector of rows
                    theMetaData = theResult.getMetaData( );
                    int columnCount = theMetaData.getColumnCount( );
                  theDisplay.setText("");
                  while (theResult.next( )) // move to next row
                   {
                      for (int i = 1; i <= columnCount; i++)
                        {
                              String colValue = theResult.getString(i); // get column i, for current row
                              if (colValue == null) colValue = "";
                              theDisplay.append(colValue + ";");
                      }
                      theDisplay.append("\n");
                  }
                } catch (Exception e)
                {   handleException(e);   }
        }
```

SQL in Program Code (PHP)

```
$query = "select * from s";
$result = mysql_query($query);  // $result is an array of arrays (rows)
$num_results = mysql_num_rows($result);
echo '<P><H2>Number of records found: '.$num_results.'</H2></P>';
echo '<TABLE BORDER>';
for ($i=0; $i<$num_results; $i++)
{
                $row = mysql_fetch_array($result); // $row is an array of columns
                echo '<TR>';
                echo '<TD>'.htmlspecialchars(stripslashes($row['sid'])).'</TD>';
                echo '<TD>'.htmlspecialchars(stripslashes($row['sname'])).'</TD>';
                echo '<TD>'.htmlspecialchars(stripslashes($row['city'])).'</TD>';
                echo '</TR>';
}
echo '</TABLE>';
```

References

- SQL for SMARTIES; Celko, J.; 1-55860-323-9
- SQL Queries: 200+ Queries to Challenge you by Swaroop Kallakuri
- Sql Guide (Quick Study: SQL) by Inc. BarCharts
- SQL Practice Problems: 57 beginning, intermediate, and advanced challenges for you to solve using a “learn-by-doing” approach by Sylvia Moestl Vasilik
- SQL in 10 Minutes a Day, Sams Teach Yourself by Ben Forta

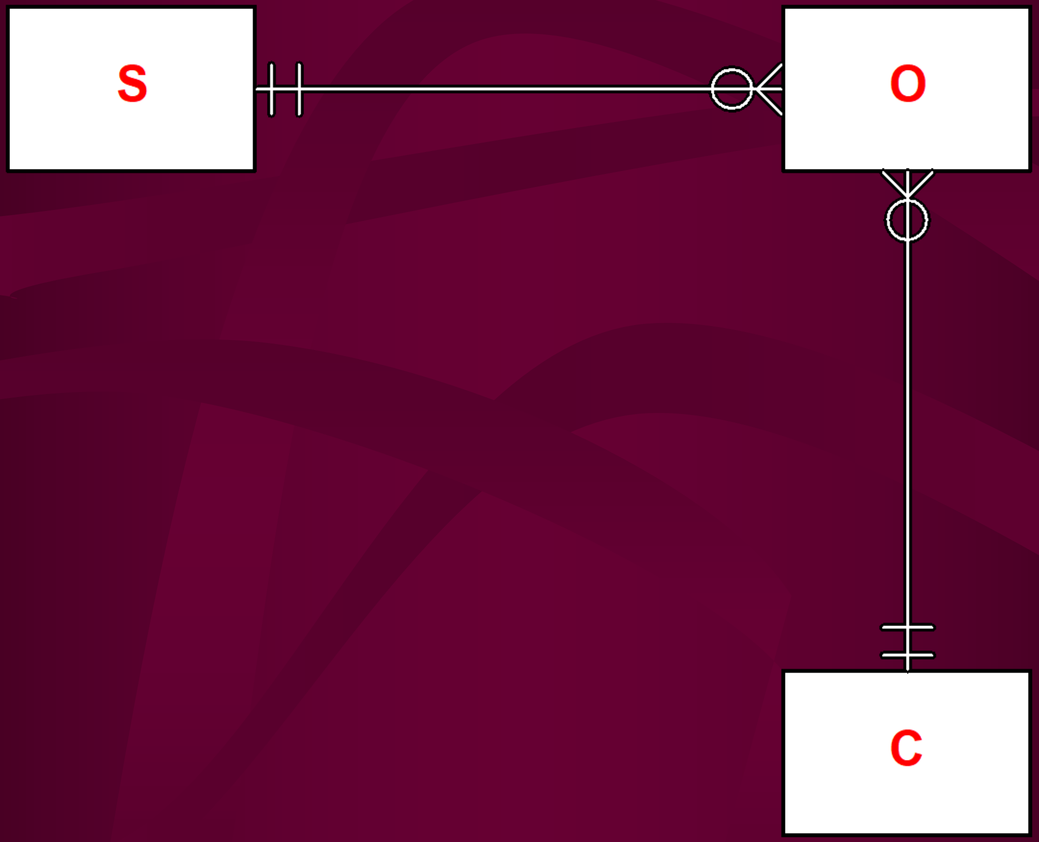
Copyright Dan Brandon

Homework

- Textbook Chapter 8
- More SOC queries (16 thru 24)
- Access model for SOC queries of last lesson
  - SOC.mdb - download from course web site- see next slide
    - Do "save as" first
- <u>SQL Test coming soon</u>

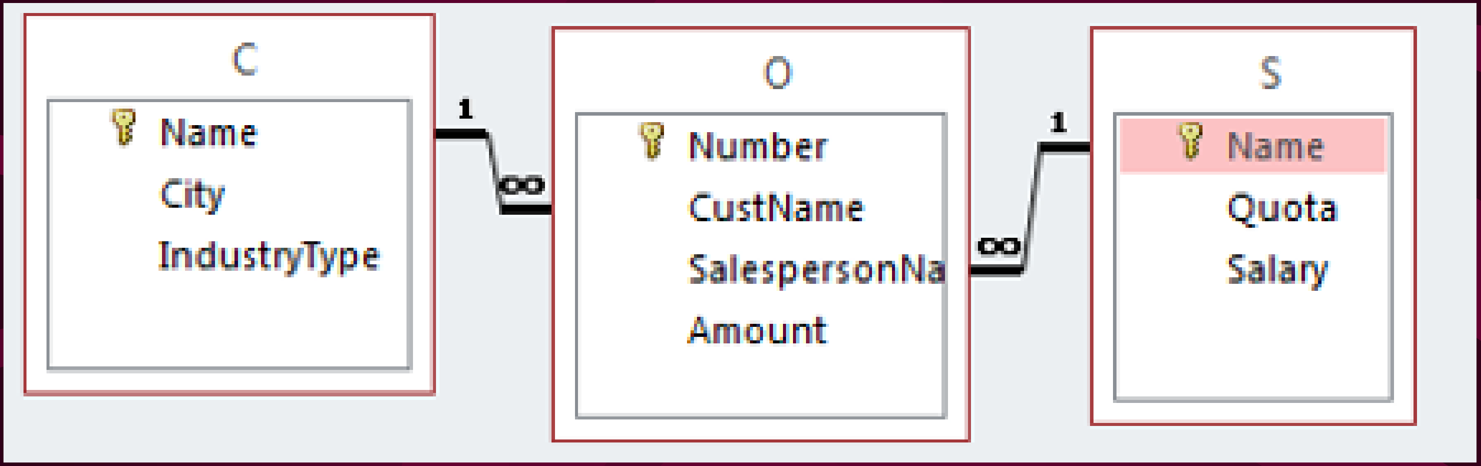
Copyright Dan Brandon

SOC E-R Diagram

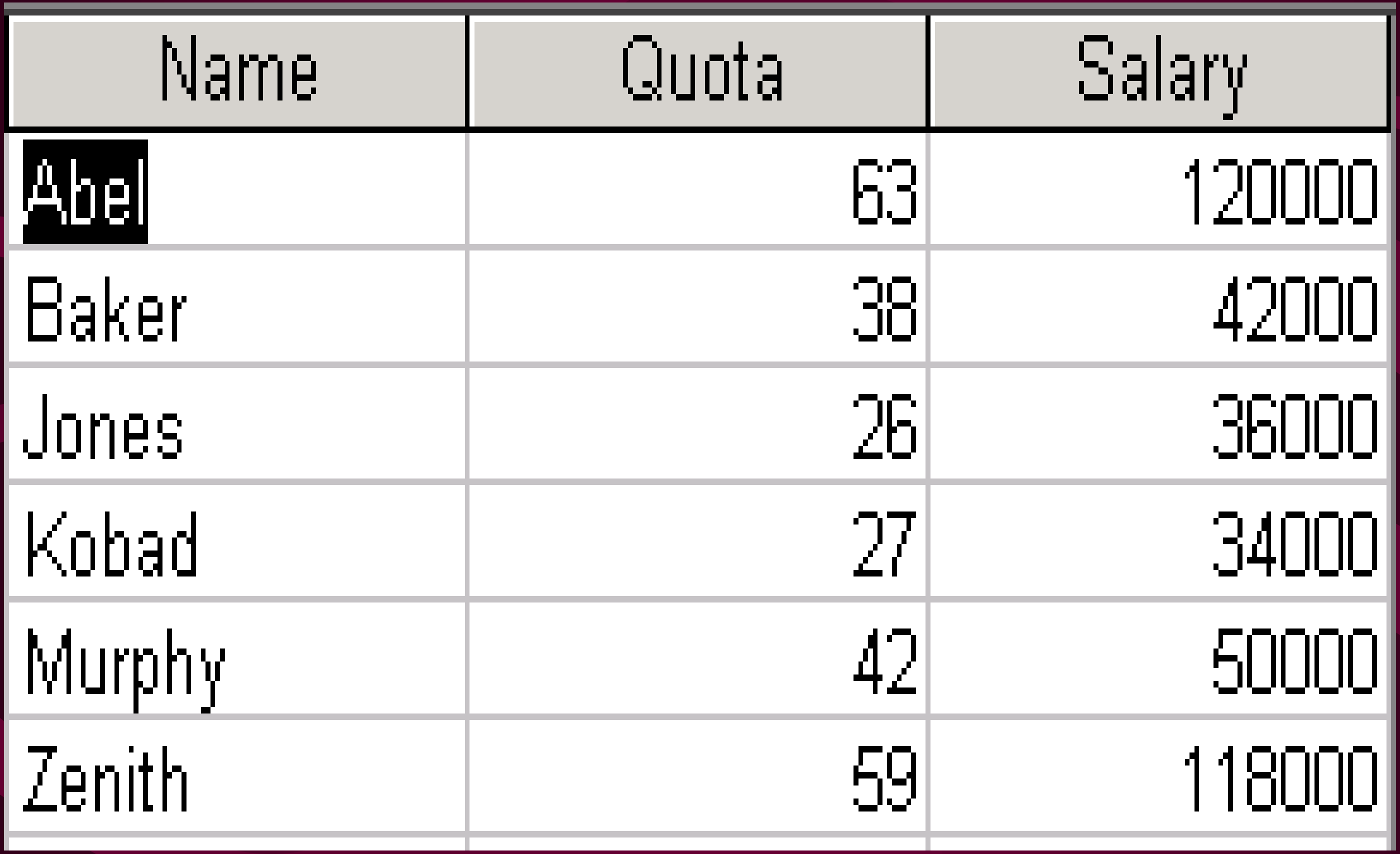


Copyright Dan Brandon

SOC Access Model



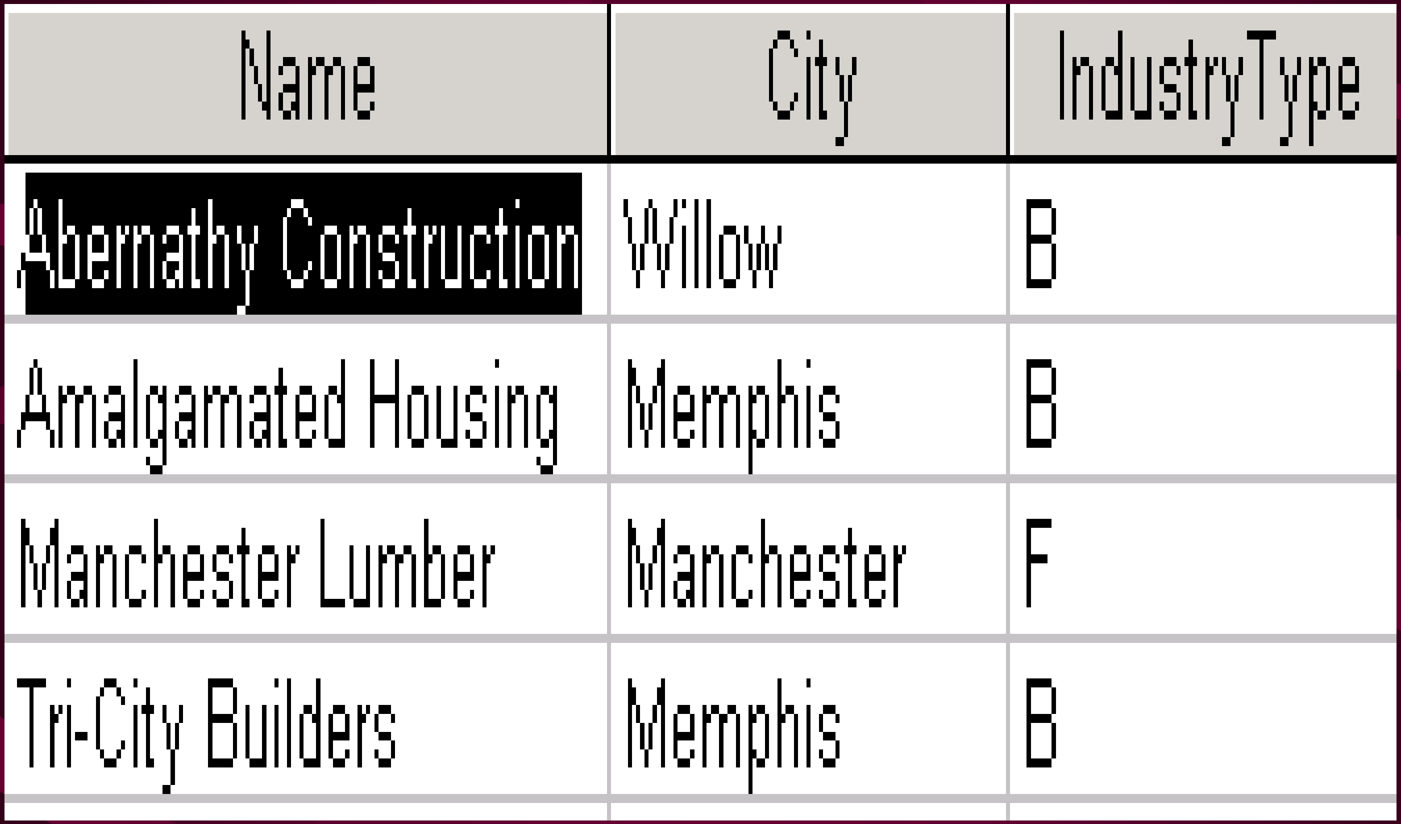
S (Salesperson Table)



Name	Quota	Salary
Abel	63	120000
Baker	38	42000
Jones	26	36000
Kobad	27	34000
Murphy	42	50000
Zenith	59	118000

Copyright Dan Brandon

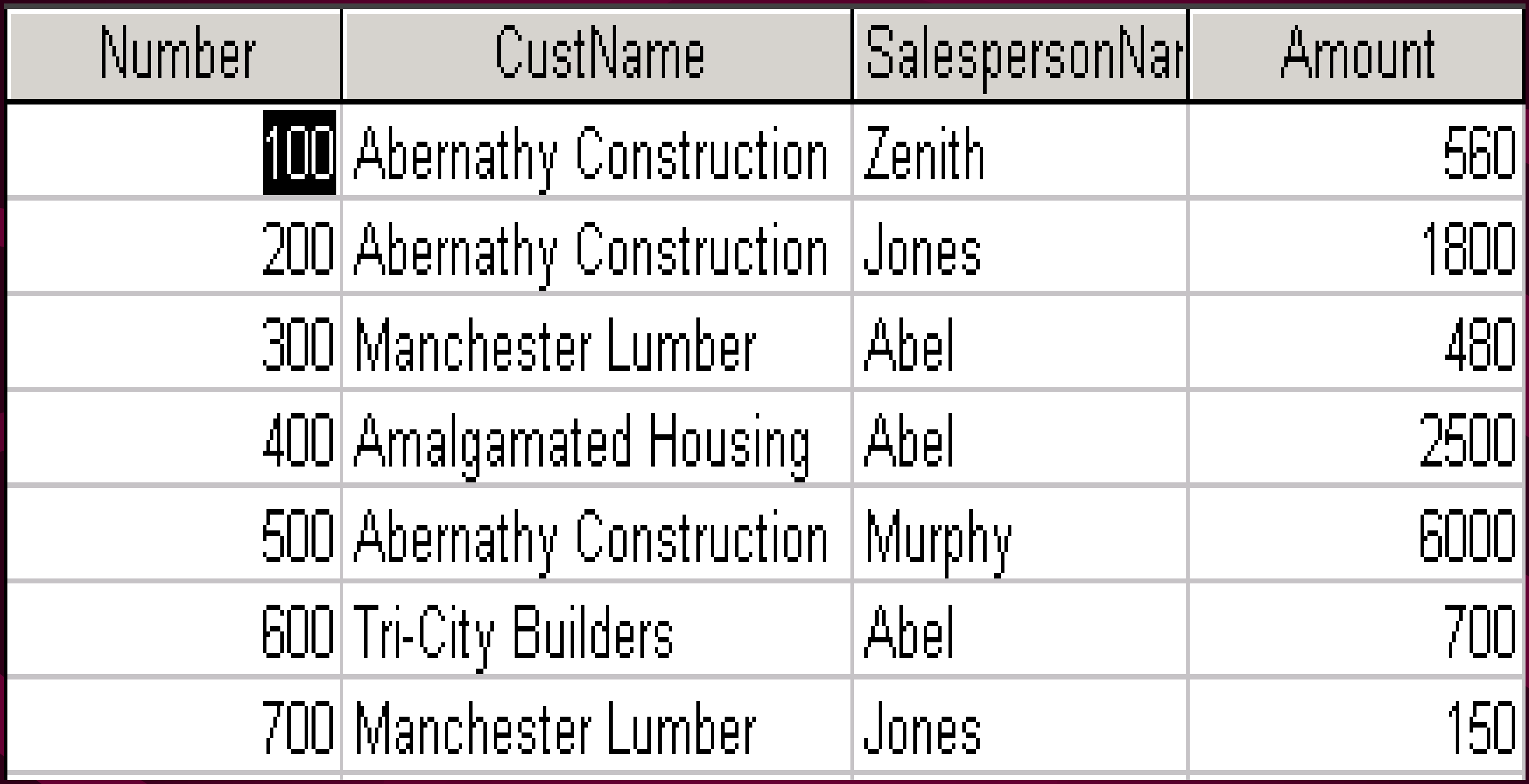
C (Customer Table)



Name	City	IndustryType
Abernathy Construction	Willow	B
Amalgamated Housing	Memphis	B
Manchester Lumber	Manchester	F
Tri-City Builders	Memphis	B

Copyright Dan Brandon

O (Order Table)



Number	CustName	SalespersonNar	Amount
100	Abernathy Construction	Zenith	560
200	Abernathy Construction	Jones	1800
300	Manchester Lumber	Abel	480
400	Amalgamated Housing	Abel	2500
500	Abernathy Construction	Murphy	6000
600	Tri-City Builders	Abel	700
700	Manchester Lumber	Jones	150

Copyright Dan Brandon

SQL Homework Queries

- 16. Show the names and quota percentages of salespeople who have an order with ABERNATHY CONSTRUCTION, in descending order of quota percentage (use a subquery).
- 17. Show the names and quota percentages of salespeople who have an order with ABERNATHY CONSTRUCTION, in descending order of quota percentage (use a join).
- 18. Show the quota percentages of salespeople who have an order with a customer in MEMPHIS (use a subquery).
- 19. Show the quota percentages of salespeople who have an order with a customer in MEMPHIS (use a join).
- 20. Show the industry type and names of the salespeople of all orders for companies in MEMPHIS.
- 21. Show the names of salespeople along with the names of the customers which have ordered from them. Include salespeople who have had no orders. Use Microsoft Access notation.
- 22. Show the names of salespeople who have two or more orders.
- 23. Show the names and quota percentages of salespeople who have two or more orders.
- 24. Show the names and quota of salespeople who have an order with all customers.

Copyright Dan Brandon