Relationships

- rep by lines ; embedded in implicit business rules (policy/procedure); lines that connect entity’s by their relationship to each other; bi-directional; always read both ways
- *anomalies*
- inner markings on lines, tells participation aka the minimum
- outer markings on lines, tells pardonality aka the maximum
- inner marking letter O for Optional aka don’t have to use it
- markings define constraints on the relationship
- 1:M and M:M

Attributes of Relationships

- some attributes not associated with entities but with relationships
- attribute off relationship, not entity
- breaks off into own entity by dotted line
- name is verb defining relationship
- orig participating entities do not get FKs anymore when this is an option // breaking apart relationship into smaller w intermediary entity
- only exist in Degree 2 and > Degree 2
- Degree 2 = M:M
- > Degree 2 = complex relations w higher degree than 2
- degree depends on num of participating entities in relationship
- ex. unary, binary, ternary, quaternary… n-ary relationships
- ex. ternary relationship – one-to-one-to-many (1:1:M), num (range) of poss occurrences of entity type in an n-ary relationship when other (n-1) values are fixed

Multiplicity

- num (range) of possible occurrences of an entity type that may relate to a single occurrence of an associated entity type through a particular relationship
- reps business rules est by user or company
- **cardinality** – max num of poss relationships
- **participation** – min num of poss relationships
- when reading, read inner then outer markings, sometimes inner not nec

PK and FK

- tell you how tables are related

FK is a PK of another table; records 1:M relationship

PK must exist before FK defined (*referential integrity*)

FK does not have to keep PK name (stored in separate tables so can have diff col name) but if diff need to be similarly recognizable

CREATING TABLES

Step #1 Name Table

Step #2 List Attributes, DataTypes, Links

Step #3 State PK

- only state NOT NULL once in PK in its own table, so you don’t have to re-define

Step #4 State FK

- FK / REFERENCES links table back

```
CREATE TABLE t1name (
        PK1name         CHAR (#) NOT NULL,
        Attname         VARCHAR (#),
        Attname         DECIMAL (#,#)
        PRIMARY KEY(PK1name));
```

```
CREATE TABLE t2name (
        PK2name         CHAR (#) NOT NULL,
        Attname         VARCHAR (#),
        Attname         DECIMAL (#,#),
        PRIMARY KEY(PK2name),
        FOREIGN KEY(PK1name) REFERENCES t1name(PK1name));

CREATE TABLE twitterUser (
        acctno  VARCHAR(30) NOT NULL,
        name    VARCHAR(2),
        sex     VARCHAR(4),
        PRIMARY KEY(acctno));

CREATE TABLE follow (
        userID  VARCHAR(30) NOT NULL,
        foleeID VARCHAR(30),
        PRIMARY KEY(userID, foleeID),
        FOREIGN KEY(userID) REFERENCES twitterUser(acctno),
        FOREIGN KEY(foleeID) REFERENCES twitterUser(acctno));
```

SUBQUERIES – NESTED

- **INNER SELECT (sub-select)** is used in the *outer* statement to help determine contents of the final result
    - used in WHERE / HAVING clauses of an *outer* SELECT statement – **subquery or nested query**
- may be in INSERT, UPDATE, DELETE statements
- 3 types:
- (1) *scalar subquery* returns single value
- (2) *row subquery* returns multi cols but only 1 row ; when row value constructor is needed in predicates
- (3) *table subquery* returns 1+ cols and multi rows ; when a table is needed (IN())
- think of as producing a temp table w results that can be accessed & used by the outer statement
- can be used immediately after relational operator (=, >, <, <=, >=, <>) in WHERE/HAVING clause

```
        SELECT staffno, fname, lname, position, salary–(SELECT AVG(salary) FROM staff) AS salDiff)
        FROM staff
        WHERE salary > (SELECT AVG(salary) FROM Staff);

        …outer query reduced to…
        SELECT staffno, fname, lname, position, salary-17000 AS salDiff
        FROM Staff
        WHERE salary > 17000;

        #List properties that are handled by staff who work in branch at ‘163 Main St.’
        SELECT propNo, street, city, postcode, type, rooms, rent
        FROM propRent
        WHERE staffNo IN (SELECT  staffNo
                          FROM Staff
                          WHERE branchNo = (SELECT branchNo
                                            FROM branch
                                            WHERE street = ‘163 Main St’));
```

JOINING TABLES

- **Cartesian Product of Sets**
- *cartesian product* - set of all ordered pairs, where 1st element is member of D1 & 2nd is member of D2

  D1 = {2, 3}
  D2 = {1. 3. 5}
  so D1 x D2 = {(2, 1), (2, 3), (2, 5), (4, 1), (4, 3), (4, 5)}

SIMPLE JOIN

- calls 2 tables in FROM clause
- use *aliases* for multi-table queries
- must define every item used in FROM clause
- rem: PK and FK do not have to have same name

```
SELECT *
FROM stock s, nation n
WHERE s.natcode = n.natcode;
```

- alternative simple joins:

```
FROM client c JOIN viewing v ON c.clientNo = v.clientNo
FROM client JOIN viewing USING clientNo
FROM client NATURAL JOIN viewing
```

VARIOUS JOINS

- cols merge by name of join (R JOIN lists new cols on R; L JOIN lists new col on L)
- outer joins are less common bc usually want to match records
- to match records use inner joins
- **3-TABLE JOIN**

```
SELECT b.branchNo, b.city, s.staffNo, fname, lname, propNo
FROM
        branch b,
        staff s,
        propRent p
WHERE b.branchNo = s.branchNo
        AND s.staffNo = p.staffNo
ORDER BY b.branchNo, s.staffNo, p.propNo;

…same as…
FROM (branch b JOIN staff s USING branchNo) AS bs
                JOIN propRent p USING staffno
```

- INNER JOIN

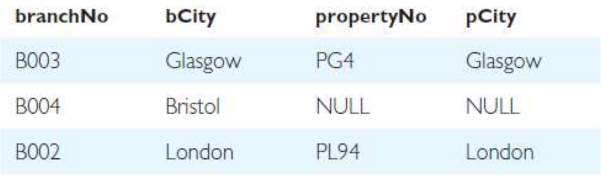
- only includes matched rows

- **OUTER JOINS**
- includes matched & unmatched rows aka NULLs
- retains rows that do not satisfy the JOIN condition
- see pg 172 in txt
- 3 types:

- (1) **LEFT JOIN**
- includes not only rows that have matching condition, but also rows of the first (left) table that are unmatched with rows from the second (right) table
- cols from second table are filled with NULLs

# List all branches and any properties that are in the same city

```
SELECT b.*, p.*
FROM branch1 b LEFT JOIN propForRent1 p ON b.bCity = p.pCity;
```

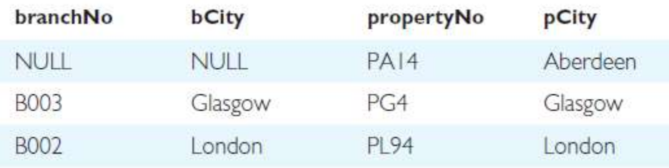


branchNo	bCity	propertyNo	pCity
B003	Glasgow	PG4	Glasgow
B004	Bristol	NULL	NULL
B002	London	PL94	London

- (2) **RIGHT JOIN**
- includes rows that have met the condition, but also rows of the 2nd (right) table that are unmatched with rows from the 1st (left) table
- cols from 1st table are filled with NULLs

# List all properties and any branch that are in the same city

```
SELECT b.*, p.*
FROM branch1 b RIGHT JOIN propForRent1 p ON b.bCity = p.pCity;
```

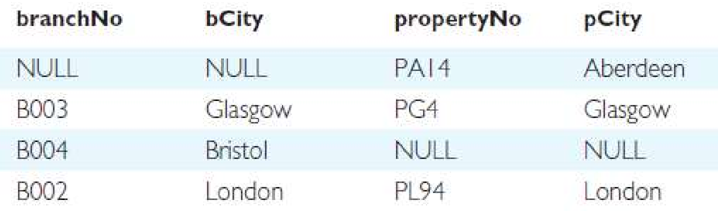


branchNo	bCity	propertyNo	pCity
NULL	NULL	PA14	Aberdeen
B003	Glasgow	PG4	Glasgow
B002	London	PL94	London

- (3) **FULL JOIN**

# List branches and properties that are in the same city along with any unmatched branches or properties

```
SELECT b.*, p.*
FROM branch1 b FULL JOIN propForRent1 p ON b.bCity = p.pCity;
```



branchNo	bCity	propertyNo	pCity
NULL	NULL	PA14	Aberdeen
B003	Glasgow	PG4	Glasgow
B004	Bristol	NULL	NULL
B002	London	PL94	London

SET OPERATIONS

- combine results of 2+ queries into a single result table
- **UNION**
- returns table containing all rows in either A or B or both
- ex. managers who are either Female **OR** below 40

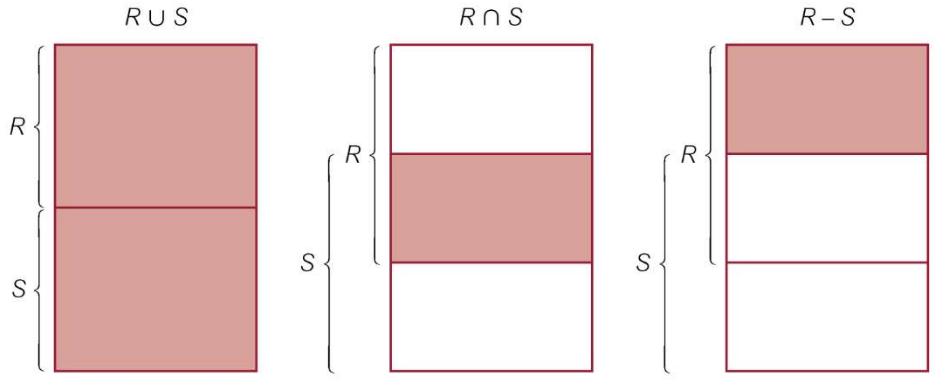
```
(SELECT city FROM branch
WHERE city IS NOT NULL)
UNION
(SELECT city FROM propRent
WHERE city IS NOT NULL);
        (query1)
        UNION
        (query2);
```

- **INTERSECTION**
- returns table containing all rows common to both A and B; matches both conditions
- use in WHERE clause
- ex. managers who are both female **AND** below 40

```
SELECT DISTINCT city FROM branch
WHERE city IN
(SELECT city FROM propRent);
```

- **DIFFERENCE**
- returns table containing all rows in A but not B
- use in WHERE clause
- ex. managers who are female **BUT NOT (/DO NOT)** below 40

```
SELECT city FROM branch
WHERE city NOT IN
(SELECT city FROM propRent);
```



(a) Union (b) Intersection (c) Difference

VIEWS

- **Base Table :** name table corresponding to entity in conceptual schema, where rows are physically stored in db
- **View :** dynamic result of 1+ relational operations operating on base table to produce another table; does not necessarily exist in db but can be produced when called, virtual table
- dynamic meaning that changes made to base tables that affect view columns are immediately reflected in the VIEW; synced
- only definition of the view is stored, not the result
- query exactly as if a table
- SELECT to return underlying base tables to view

```
CREATE VIEW tname (att1, att2, att3, att4, att5) AS
        SELECT (b.att1, a.att1, a.att2, a.att3, b.att2, b.att3*a.att1)
        FROM table1 a, table2 b
        WHERE a.PK = b.FK;
```

GROUP BY

- clause to create groups
- form groups of columns in SELECT then ORDER BY sorts them
- can use multi cols to create groups at more granular level // intermediate rollup for subtotal of each group
- an aggregate function, aka only returns 1 row
- integrated with SELECT – each item in SELECT must be listed in GROUP BY clause; also, any attribute that's aggregate function is applicable and constants
- stats at *most aggregate level*
- WHERE comes first b/c groups are formed from remaining rows satisfying the predicate; filter rows first, then make groups

- how many x in each x; total x paid by each x; how many x in each x by x
- ex. how many in total
- ex. with 1 table (num of x in each y by gender)

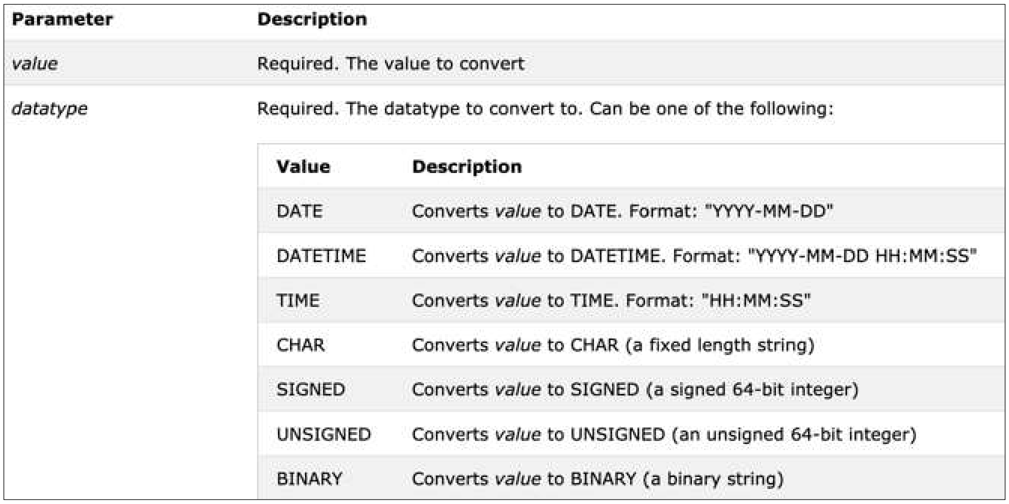
```
SELECT num,
        COUNT(xno) AS totalx,
        SUM(y) AS totally
FROM table
GROUP BY num;
```

SQL SCALAR OPERATORS

CAST

- data type convert function
- use to separate Date & Time in column; convert to integer
- can tweak to make output look as desired

```
CAST(value AS datatype)
```



Parameter	Description
value	Required. The value to convert
datatype	Required. The datatype to convert to. Can be one of the following:

Value	Description
DATE	Converts *value* to DATE. Format: "YYYY-MM-DD"
DATETIME	Converts *value* to DATETIME. Format: "YYYY-MM-DD HH:MM:SS"
TIME	Converts *value* to TIME. Format: "HH:MM:SS"
CHAR	Converts *value* to CHAR (a fixed length string)
SIGNED	Converts *value* to SIGNED (a signed 64-bit integer)
UNSIGNED	Converts *value* to UNSIGNED (an unsigned 64-bit integer)
BINARY	Converts *value* to BINARY (a binary string)

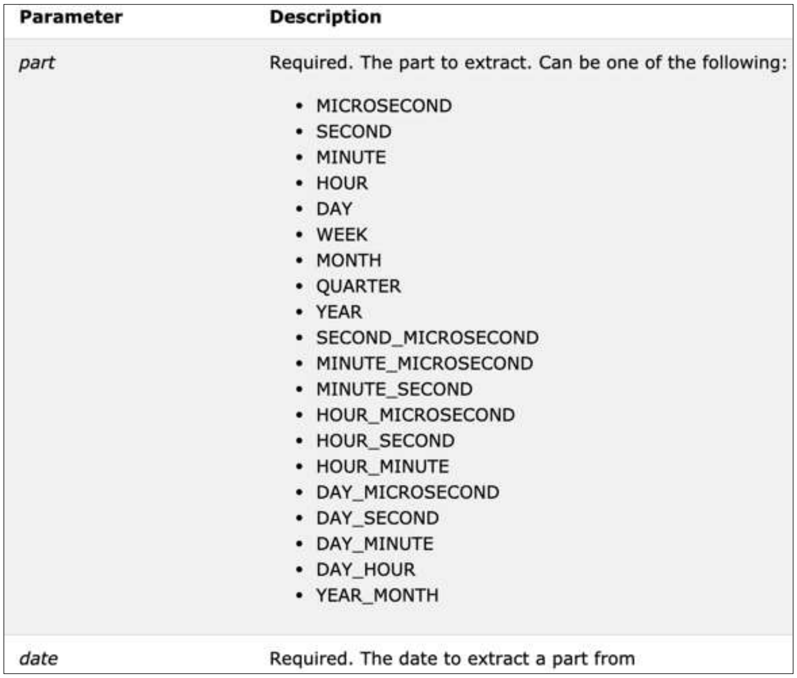
```
SELECT clientNo, branchNo, staffNo
        CAST(dateJoined AS DATE) AS joinDate,
        CAST(dateJoined AS TIME) AS joinTime
 FROM registration;

SELECT orderNo, prodCode,
        CAST(price AS SIGNED)
 FROM registration;
```

EXTRACT

- date & time function
- returns value of specified file from a datetime or interval value
- similar to DATEPART( ) function
- use to separate year, month, day

```
EXTRACT(part FROM date)
```



Parameter	Description
part	Required. The part to extract. Can be one of the following: • MICROSECOND • SECOND • MINUTE • HOUR • DAY • WEEK • MONTH • QUARTER • YEAR • SECOND_MICROSECOND • MINUTE_MICROSECOND • MINUTE_SECOND • HOUR_MICROSECOND • HOUR_SECOND • HOUR_MINUTE • DAY_MICROSECOND • DAY_SECOND • DAY_MINUTE • DAY_HOUR • YEAR_MONTH
date	Required. The date to extract a part from

```
SELECT  EXTRACT(year FROM DOB) AS DOBYear,
        EXTRACT(month FROM DOB) AS DOBMonth,
        EXTRACT(day FROM DOB) AS DOBDay
FROM staff;
```

CHARACTER STRING MANIPULATION

- **LOWER( ) … UPPER( )**

- converts lower-/upper-case letters to upper-/lower-case

```
# List staff’s fname in lowercase and last name in upper
SELECT LOWER(fname), UPPER(lname)
FROM staff;
```

- **TRIM( )**

- function returns a string after removing all prefixes/suffixes from given string; chop off beginning or end of something; goes in SELECT clause

TRIM([{BOTH | LEADING | TRAILING} [REMSTR] FROM] str)

```
# to trim at beginning of name
SELECT TRIM(LEADING ‘ ‘ FROM sktfirm)
FROM stock;
# Trim space at end of name
SELECT TRIM(TRAILING ‘ ‘ FROM sktfirm)
FROM stock;
# Trim space at both beginning / end of name
SELECT TRIM(BOTH ‘ ‘ FROM stkfirm)
FROM strock;
```

- **POSITION( )**
- returns position of a substring w/in a string; every character has its own unique position; position of a string; starts with 1, no positional index
- if returns 1st position that it finds, not all of them, or returns 0 of that character if it doesn't exist; only returns position of 1st occurance
- instead of num position, returns string in that position

```
        POSITION(substr IN string)

        SELECT POSITION(‘ ‘ IN guestname)
        FROM guest;
```

- **SUBSTRING( )**
- extracts substring from a string (starting at any position)

```
        SUBSTRING(string, start, length)
```

# extract fnamein gname of guest table, using substring and position together
- ending with the space is what separates the name, is what returns just first name

```
        SELECT SUBSTRING(gname, 1, POSITION(‘ ‘ IN gname))
        FROM guest;
```

# extract first 3 characters in gname of guest table

```
        SELECT SUBSTRING(gname, 1, 3)
        FROM guest;
```

- **LEFT( ) … RIGHT( )**
- function extracts a num of characters from a string (starting from left/right)

```
        LEFT(string, numofChars)
        RIGHT(string, numofChars)
```

LOGICAL OPERATION

IF

- function takes 3 expressions… if 1st is true, not zero & not NULL, returns 2nd expression… otherwise, returns 3rd
- depending on context, returns either numeric or string value

```
        IF(expression, exproTrue, exprFalse);
        # Categorize staff to high salary vs low salary by salary at 20,000
        SELECT staffno, salary,
                IF(salary > 20000, ‘High’, ‘Low’) AS salaryLevel
        FROM staff;
```

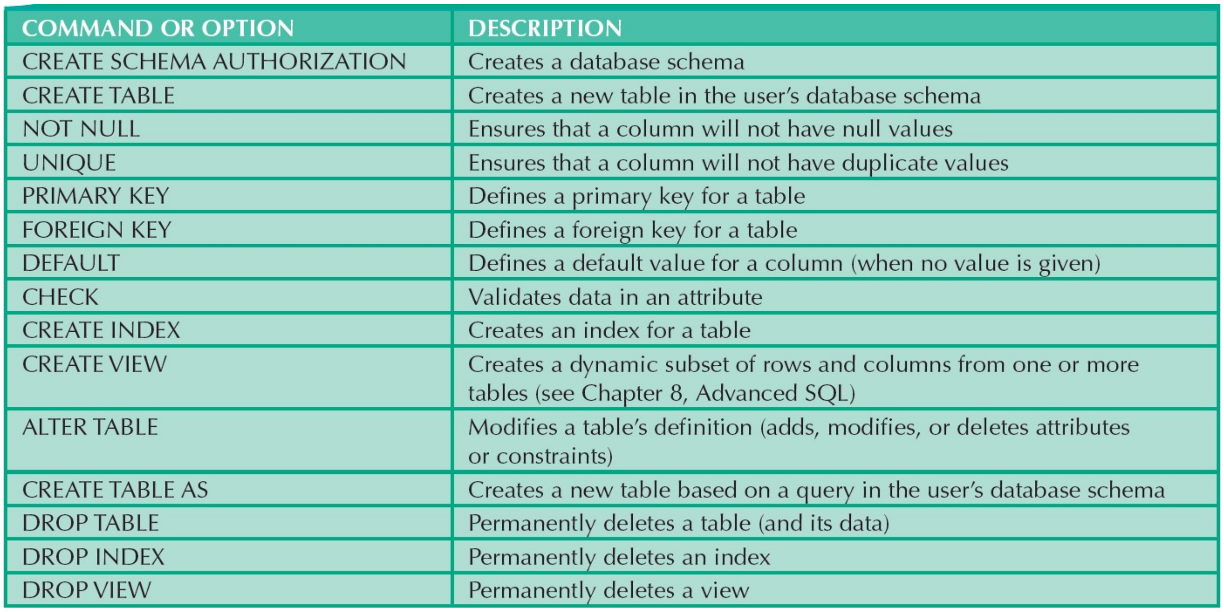
CASE FUNCTION

- goes thru condition & returns a value when the 1st condition is met (like an IF-THEN-ELSE statement)… once true, will stop reading & return result

```
        CASE
                WHEN condition1 THEN result1
                WHEN condition2 THEN result2
                …
                WHEN conditionN THEN result
                ELSE result
        END;
```

```
#Cat staff to high, mid and low salary level
SELECT staffNo, salary
CASE
        WHEN salary > 20000 THEN 'high'
        WHEN salary between 10000 AND 20000 THEN 'mid'
        ELSE 'low'
END
AS salaryLevel
FROM staff;
```

DDL Commands



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

Cengage Learning © 2015

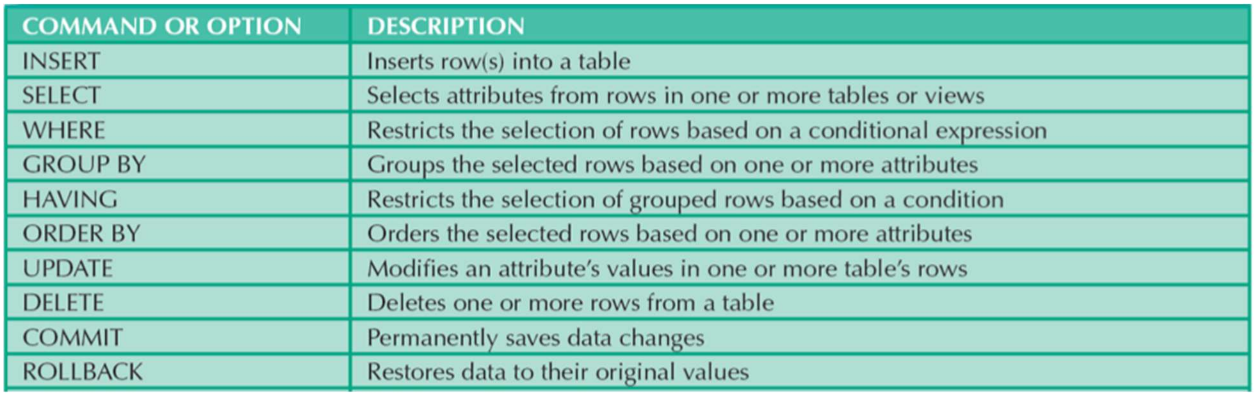
NOT NULL – cannot have a null value for that attribute, fields cannot be blank

ALLOWABLE DATA TYPES IN SQL

Numeric	integer	A 31-bit signed binary value
	smallint	A 15-bit signed binary value
	float(*p*)	A scientific format number of *p* binary digits precision
	decimal(*p,q*)	A packed decimal number of *p* digits total length; *q* decimal places to the right of the decimal point may be specified
String	char(*n*)	A fixed length character string of *n* characters
	varchar(*n*)	A variable length character string up to *n* characters
	text	A variable-length character string of up to 65,535 characters
Date/time	date	Date in the form *yyyymmdd*
	time	Time in the form *hhmmss*
	timestamp	A combination of date and time to the nearest microsecond
	time with time zone	Same as time, with the addition of an offset from UTC of the specified time
	timestamp w/ time zone	Same as timestamp, with the addition of an offset from UTC of the specified time

CHAR can be a number if you know you're never going to use it for mathematical reasons

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

Anything after SELECT is related to the SELECT statement; COMMIT can be automatic
*Note: DROP is a DDL command because it drops the whole table while DELETE is a DML commany because it removes only the row

INSERTING ROWS

- **VALUE :** tells which column to send new info to with order of values pairs
- if text (non-numeric literals), must be enclosed by *single* quotes
- if number (numeric literal), must *not* use quotes

```
INSERT INTO share (shrcode, shrfirm, shrprice, shrqty, shrdiv, shrpe)
VALUES ('FC', 'Freedonia Copper', 27.5, 10529, 1.84, 16);

INSERT INTO tname (col1name, col2name, col3name, col4name…)
VALUES ('newvalue', 'nv', newnum…);

INSERT INTO share
VALUES('FC', 'Freedonia Copper', 27.5, 10529, 1.84, 16);

INSERT INTO tname
VALUES('newval', 'nv', num…);
```

NULL

- shows absence of value & is not same as 0 or spaces, which are values ; Representative of an attribute that is currently unknown or not application for tuple; deals with incomplete or exceptional data
- PKs cannot be NULL, they must exist
- an 'entity integrity constraint'

Name	UID	Email	SSN	GPA	Degree	Grad_Yr
John Smith	110718	johnsmith@lsu.edu	12345	3.7894	IS	2014
XXX	110999	NULL	NULL	3.8000	IS	2014

DATA MINING

- SAS is a leader vendor
- challenge is finding suitable data to mine
- data quality & consistency is a pre-req for mining to ensure accuracy of predictive models
- data warehouses good for providing data for mining & have capability to go back to data source
- rem: GIGO, garbage in and garbage out
- operationally intensive so using a lot of house power therefore don't want to use against OS so want to do tests on different servers

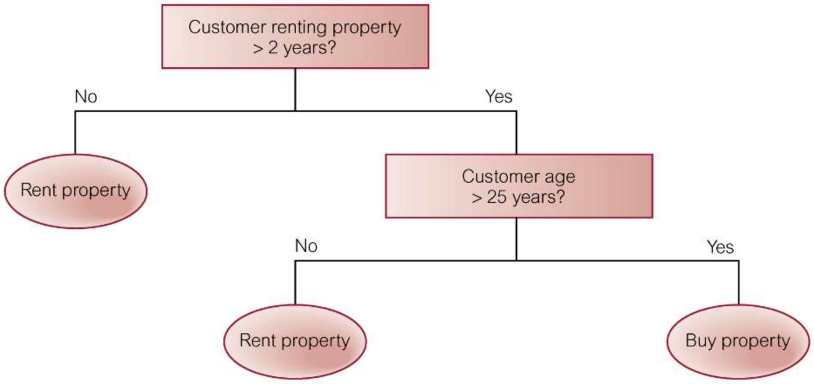
Operations	Data mining techniques
Predictive modeling	Classification
	Value prediction
Database segmentation	Demographic clustering
	Neural clustering
Link analysis	Association discovery
	Sequential pattern discovery
	Similar time sequence discovery
Deviation detection	Statistics Visualization

PREDICTIVE MODELING

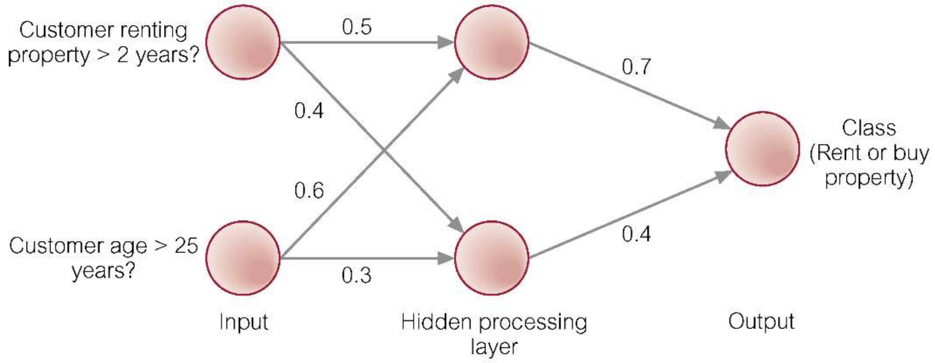
- Similar to human learning
- uses observations to form a model of the important characteristics of some phenomenon
- uses generalizations of ‘real world’ & ability to fit new data into a general framework
- analyze db to determine essential characteristics (model) about the data set
- Model is developed using a *supervised learning* approach, which has two phases: training and testing.
    - *Training* builds a model using lg sample of historical data (training set)
    - T*esting* is trying out the model on previously unseen data to determine models accuracy & physical performance characteristics
- customer retention management, credit approval, cross selling, direct marketing
- 2 predictive modeling techniques: Classification or Value prediction
    - distinguished by nature of variable being predicted

- (1) CLASSIFICATION

- to establish specific predetermined class for each record in a db from a finite set of possible, class values
- Fit records into group that is pre-determined
- 2 special classifications:
    - Tree induction



- Neural induction



- (2) VALUE PREDICTION

- to estimate a *continuous numeric value* associated with a db record
- uses linear regression & nonlinear regression
- linear regression attempts to fit a straight line through a plot of the data where the line is best rep of the avg of all observations at that point in the plot // Linear Regression fits straight line to dataset to extract info for future observations
- credit card fraud detection or target mailing list identification, send mail to people closest to linear line

DATABASE SEGMENTATION

- to partition a db into an unknown number of segments (clusters) of similar records // partition db into clusters
- uses *unsupervised learning* to discover homogeneous sub-populations (from clusters with similar traits) in a db to improve accuracy of the profiles // don't know how many clusters will form (unsupervised)
- less precise than other operations thus less sensitive to redundant and irrelevant features
- customer profiling, direct marketing, cross selling

LINK ANALYSIS

- to establish links (associations) between records, or sets of records, in a db // est specific association b/t records
- 3 specializations: (1) associations discovery (2) sequential pattern discovery (3) similar time sequence discovery
- product affinity analysis, direct marketing, stock price movement
- do have a relationship, either direct or inverse but there is a relationship

- (1) ASSOCIATIONS DISCOVERY:

- finds items that imply the presence of other items in the same event
- Affinities between items are represented by association rules
    - e.g. 'When a customer rents property for more than 2 years and is more than 25 years old, in 40% of cases, the customer will buy a property. This association happens in 35% of all customers who rent properties'

- (2) SEQUENTIAL PATTERN DISCOVERY:

- find patterns bt events where presence of one set of items is followed by another set of items in a db of events over a pd of time
- understand long term customer buying behavior

- **(3) SIMILAR TIME SEQUENCE DELIVERY:**

- finds links bt 2 sets of data that are time-dependent & is based on the degree of similarity bt patterns that both times series demonstrate
- w/in 3 mo of buying property, new home owners will purchase goods: cookers, freezers, washing machines

DEVIATION DETECTION

- often source of discovery bc identifies outliers, which express deviation from some previously known expectation and norm
- performed using statistics & data visualization or as by-product of data mining
- fraud detection in credit cards & ins claims, quality control, defects tracing