Ceng 302
Database Management Systems

SQL: Structured Query Language

Prof. Dr. Adnan YAZICI

Department of Computer Engineering,

Middle East Technical University

(Fall 2021)

Objectives

- Introduction to SQL (review)
- SQL Commands - Data Definition Language (DDL) - examples
- Data Manipulation Language (DML) - examples
- Stored Procedures
- Triggers
- Views
- Indexes and B+Tree
- Recursive SQL
- Integrity constraint
- DCL – Data Control Language (Authorization)

History of SQL

- SEQUEL: Structured English QUEry Language; part of SYSTEM R, 1974
- SQL/86: ANSI & ISO standard
- SQL/89: ANSI & ISO standard
- SQL/92 or SQL2: ANSI & ISO standard
- SQL3: in the works...
- SQL is supported by ORACLE, SYBASE, INFORMIX, IBM DB2, SQL SERVER, OPENINGRES,POSTGRESQL, MYSQL...

SQL and Relational Calculus

- Although relational *algebra* is useful in the analysis of query evaluation, SQL is actually based on a different query language: ***relational calculus***
- There are two relational calculi:
  - ***Tuple*** *relational calculus* (TRC)
  - ***Domain*** *relational calculus* (DRC)
- SQL is based on the relational **tuple** calculus

Introduction to SQL

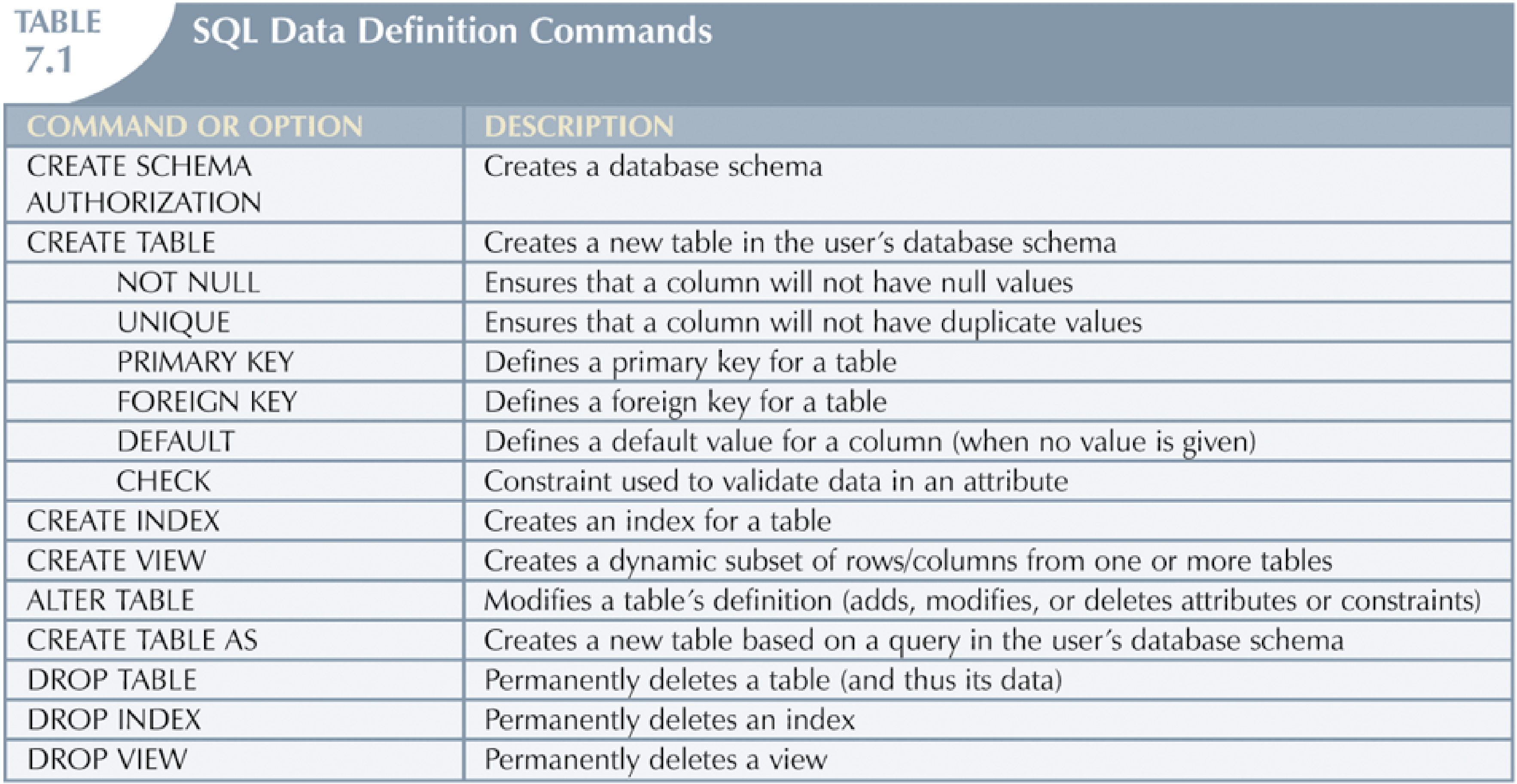

TABLE 7.1 SQL Data Definition Commands

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	Constraint used to validate data in an attribute
CREATE INDEX	Creates an index for a table
CREATE VIEW	Creates a dynamic subset of rows/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 thus its data)
DROP INDEX	Permanently deletes an index
DROP VIEW	Permanently deletes a view

Introduction to SQL (continued)

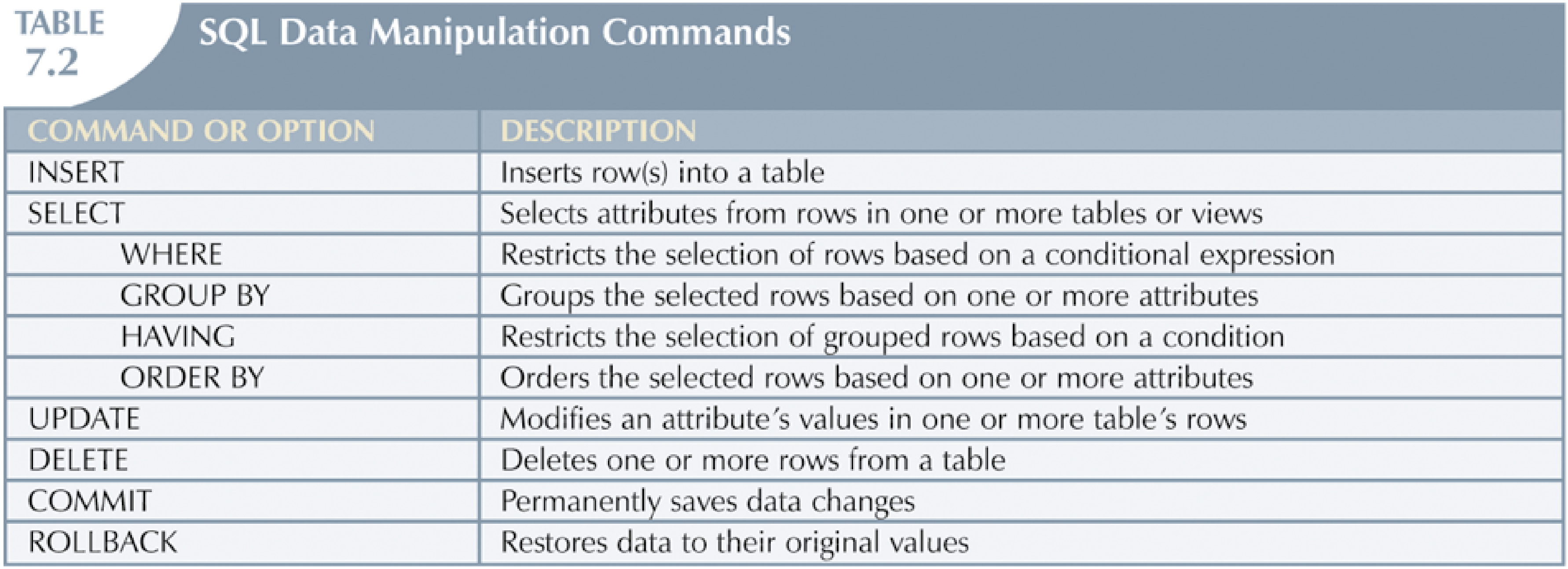

TABLE 7.2 SQL Data Manipulation Commands

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

Introduction to SQL (continued)

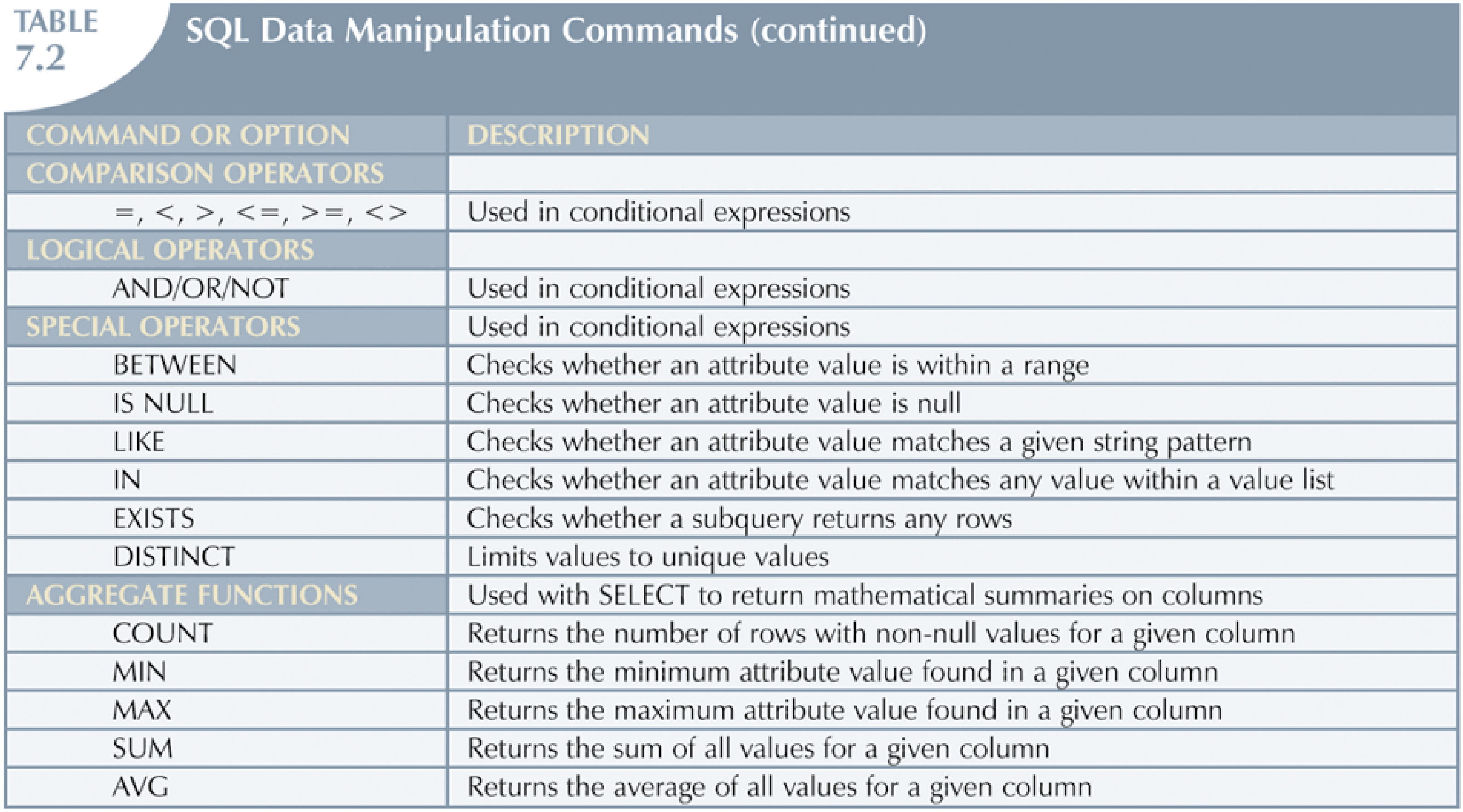

TABLE 7.2 SQL Data Manipulation Commands (continued)

COMMAND OR OPTION	DESCRIPTION
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

AIRPORT

airportcode	name	city	state

FLT-SCHEDULE

flt#	airline	dtime	from-airportcode	atime	to-airportcode	miles	price

FLT-WEEKDAY

flt#	weekday

FLT-INSTANCE

flt#	date	plane#	#avail-seats

AIRPLANE

plane#	plane-type	total-#seats

CUSTOMER

cust#	first	middle	last	phone#	street	city	state	zip

RESERVATION

flt#	date	cust#	seat#	check-in-status	ticket#

DDL - Overview

- primitive types
- domains
- schema
- tables

DDL - Primitive Types

- numeric
  - INTEGER (or INT), SMALLINT are subsets of the integers (machine dependent)
  - REAL, DOUBLE PRECISION are floating-point and double-precision floating-point (machine dependent)
  - FLOAT(N) is floating-point with at least N digits
  - DECIMAL(P,D) (or DEC(P,D), or NUMERIC(P,D)), with P digits of which D are to the right of the decimal point.

DDL - Primitive Types (cont.)

- character-string
  - CHAR(N) (or CHARACTER(N)) is a fixed-length character string
  - VARCHAR(N) (or CHAR VARYING(N), or CHARACTER VARYING(N)) is a variable-length character string with at most N characters
- bit-strings
  - BIT(N) is a fixed-length bit string
  - VARBIT(N) (or BIT VARYING(N)) is a bit string with at most N bits

DDL - Primitive Types (cont.)

- date**:** Dates, containing a (4 digit) year, month and date
  - Ex: **date** ‘2005-7-27’
- time**:** Time of day, in hours, minutes and seconds.
  - Ex: **time** ‘09:00:30’ **time** ‘09:00:30.75’
- timestamp: date plus time of day
  - Ex: **timestamp** ‘2005-7-27 09:00:30.75’
- interval**:** period of time
  - Ex: interval ‘1’ day
  - Subtracting a date/time/timestamp value from another gives an interval value
  - Interval values can be added to date/time/timestamp values

Large-Object Types

- Large objects (photos, videos, CAD files, etc.) are stored as a *large object*:
  - blob: binary large object -- object is a large collection of uninterpreted binary data (whose interpretation is left to an application outside of the database system)
  - clob: character large object -- object is a large collection of character data
  - When a query returns a large object, a pointer is returned rather than the large object itself.

DDL - Domains

- a domain can be defined as follows:

```
CREATE DOMAIN AIRPORT-CODE CHAR(3);
CREATE DOMAIN FLIGHTNUMBER CHAR(5);
```

- using domain definitions makes it easier to see which columns are related
- changing a domain definition one place changes it consistently everywhere it is used
- default values can be defined for domains
- constraints can be defined for domains

DDL - Domains (cont.)

- all domains contain the value, NULL.
- to define a different default value:

CREATE DOMAIN AIRPORT-CODE CHAR(3) **DEFAULT** ‘<literal>’;

CREATE DOMAIN AIRPORT-CODE CHAR(3) **DEFAULT** ‘niladic function’;

- literal, such as ‘???’, ‘NO-VALUE’,...
- niladic function, such as USER, CURRENT-USER, SESSION-USER, SYSTEM-USER, CURRENT-DATE, CURRENT-TIME, CURRENT-TIMESTAMP

DDL - Domains (cont.)

- a domain is dropped as follows:

  DROP DOMAIN AIRPORT-CODE **RESTRICT;**

  DROP DOMAIN AIRPORT-CODE **CASCADE;**

- restrict: drop operation fails if the domain is used in column definitions
- cascade: drop operation causes columns to be defined directly on the underlying data type

DDL - Schema

- create a schema:

```
CREATE SCHEMA AIRLINE AUTHORIZATION SMITH;
```

- the schema AIRLINE has now been created and is owner by the user “SMITH”
- tables can now be created and added to the schema

- to drop a schema:

```
DROP SCHEMA AIRLINE RESTRICT;
DROP SCHEMA AIRLINE CASCADE;
```

- restrict: drop operation fails if schema is not empty
- cascade: drop operation removes everything in the schema

DDL - Tables (cont.)

- to drop a table:

  DROP TABLE RESERVATION **RESTRICT;**

  DROP TABLE RESERVATION **CASCADE;**

- restrict: drop operation fails if the table is referenced by some view/constraint definitions
- cascade: drop operation removes referencing view/constraint definitions

DDL - Tables (cont.)

- to add a column to a table:

  ALTER TABLE AIRLINE.FLT-SCHEDULE
  ADD PRICE DECIMAL(7,2);

- if no DEFAULT is specified, the new column will have NULL values for all tuples already in the database
- to drop a column from a table

  ALTER TABLE AIRLINE.FLT-SCHEDULE
  DROP PRICE **RESTRICT (**or **CASCADE)**;

- restrict: drop operation fails if the column is referenced
- cascade: drop operation removes referencing view/constraint definitions

Constraints on a Single Relation

- **not null**
- **primary key**
- **unique**
- **check** (*P*), where *P* is a predicate

Not Null Constraint

- Declare *branch_name* for *branch* is **not null**

branch_name **char**(15) **not null**

- Declare the domain *Dollars* to be **not null**

create domain *Dollars* **numeric**(12,2) **not null**

The Unique Constraint

- **unique** (A_1, A_2, …, A_m)
- The unique specification states that the attributes

  *A*1, *A*2, … *A*m

  form a candidate key.
- Candidate keys are permitted to be null (in contrast to primary keys).

The check clause

- **check** (*P*), where *P* is a predicate

Example: Declare *branch_name* as the primary key for *branch* and ensure that the values of *assets* are non-negative.

```
create table branch
        (branch_name    char(15) NOT NULL,
         branch_city     char(30),
         assets          integer,
         primary key (branch_name),
         check (assets >= 0))
```

The check clause (Cont.)

- The **check** clause permits domains to be restricted:
  - Use **check** clause to ensure that an *hourly_wage* domain allows only values greater than a specified value.

```
create domain hourly_wage numeric(5,2)
        constraint value_test check(value > 8.00)
```

  - The domain has a constraint that ensures that the *hourly_wage* is greater than 8.00
  - The clause **constraint** *value_test* is optional; useful to indicate which constraint an update violated.

Referential Integrity in SQL – Example (Cont.)

```
create table account
   (account_number                  char(10) NOT NULL,
   branch_name                      char(15),
   balance                          integer,
   primary key (account_number),
   foreign key (branch_name) references branch )


create table depositor
   (customer_name                   char(20) NOT NULL,
   account_number                   char(10) NOT NULL,
   primary key (customer_name, account_number),
   foreign key (account_number ) references account,
   foreign key (customer_name ) references customer )
```

DDL - Tables

- to create a table in the AIRLINE schema:

```
CREATE TABLE AIRLINE.FLT-SCHEDULE
    (FLT#                       FLIGHTNUMBER NOT NULL,
    AIRLINE                     VARCHAR(25),
    FROM-AIRPORTCODE            AIRPORT-CODE,
    DTIME                       TIME,
    TO-AIRPORTCODE              AIRPORT-CODE,
    ATIME                       TIME,
    PRIMARY KEY (FLT#),
    FOREIGN KEY (FROM-AIRPORTCODE)
                    REFERENCES  AIRPORT(AIRPORTCODE),
    FOREIGN KEY (TO-AIRPORTCODE)
                    REFERENCES AIRPORT(AIRPORTCODE));
```

Relational Query Languages

- ***Query languages***: Allow manipulation and retrieval of data from a database.
- Relational model supports simple, powerful QLs:
  - Strong formal foundation based on formal logic.
  - Allows for much optimization.
- Query Languages **!=** programming languages!
  - QLs are not expected to be "Turing complete" or "computationally universal. "
  - QLs are not intended to be used for complex calculations.
  - QLs support easy, efficient access to large data sets.

Interactive DML - Overview

- select-from-where
- select clause
- where clause
- from clause
- tuple variables
- string matching
- ordering of rows
- set operations
- built-in functions
- nested subqueries
- joins
- recursive queries
- insert, delete, update

Interactive DML - select-from-where

SELECT $A_1, A_2, \ldots A_n$

FROM $R_1 , R_2 , \ldots R_m$

WHERE P

$$\pi_{A_1, A_2, \ldots A_n}(\sigma_P(R_1 \text{ x} R_2 \text{ x} \ldots \text{x } R_m))$$

- the **SELECT** clause specifies the columns of the result
- the **FROM** clause specifies the tables to be scanned in the query
- the **WHERE** clause specifies the condition on the columns of the tables in the **FROM** clause
- equivalent algebra statement:

Basic SQL Query

```
SELECT    [DISTINCT]  target-list
FROM      relation-list
WHERE     qualification
```

- *relation-list* A list of relation names (possibly with a *range-variable* after each name).
- *target-list* A list of attributes of relations in *relation-list*
- *qualification* Comparisons (Attr *op* const or Attr1 *op* Attr2, where *op* is one of (<, >, =, ≤, ≥, ≠) combined using AND, OR and NOT.
- DISTINCT is an optional keyword indicating that the answer should not contain duplicates. Default is that duplicates are *not* eliminated!

Conceptual Evaluation Strategy

- Semantics of an SQL query defined in terms of the following conceptual evaluation strategy:
  - Compute the cross-product of *relation-list.*
  - Discard resulting tuples if they fail *qualifications.*
  - Delete attributes that are not in *target-list.*
  - If DISTINCT is specified, eliminate duplicate rows.
- This strategy is probably the least efficient way to compute a query!
- An optimizer will find more efficient strategies to compute the same answers.