CHAPTER 5:

INTRODUCTION TO **S**tructured

Query

Language

Outline

- Overview of The SQL Query Language
- Data Definition Language
- Data manipulation Language
- Data Control Language

SQL-Overview

- SQL, pronounced ‘Sequel’ or simply S-Q-L, is a computer programming language that was developed especially for querying relational databases using a non-procedural approach.
- The term non-procedural means that you can extract information by simply telling the system what information is needed without telling how to perform the data retrieval. The RDBMS parses (converts) the SQL commands and completes the task.

SQL-Overview…

- Extracting information from the database by using SQL is termed ***querying*** the database.
- SQL is a language that is fairly simple to learn, in terms of writing queries but it has considerable complexity because it is a very powerful language.

SQL-Overview…

- Extracting information from the database by using SQL is termed ***querying*** the database.
- SQL is a language that is fairly simple to learn, in terms of writing queries but it has considerable complexity because it is a very powerful language.

Overview of SQL…

- SQL language is divided into four types of primary language statements:
  - **DDL (Data Definition Language)**
  - **DML (Data Manipulation Language)**
  - **DCL (Data Control Language)**
  - **TCL (Transaction Control Language)**
    - Using these statements, we can define the structure of a database by creating and altering database objects, and we can manipulate data in a table through updates or deletions.
    - We also can control which user can read/write data or manage transactions to create a single unit of work.

SQL DDL

DDL

- **Data Definition Language** (DDL) statements are used to define the database structure or schema. Some examples:
  - **CREATE -** to create objects in the database
  - **ALTER -** alters the structure of the database
  - **DROP -** delete objects from the database

Identifiers

- May contain A-Z, a-z, 0-9, _
- No longer than 128 characters(SQL Server 2012)
- Start with letter
- Cannot contain spaces

Data Types

Data Type	Declaration	Example
Boolean (TRUE/FALSE)	BOOLEAN	
character	CHAR	Bno CHAR(4)
	VARCHAR	Name VERCHAR(15)
exact numeric	NUMERIC	
	DECIMAL, DEC	salary DECIMAL(7,2)
	INTEGER, INT	
	SMALLINT	NoRoom SMALLINT

Data Types

Data Type	Declaration	Example
aprox numeric (e)	FLOAT	
	REAL	
	DOUBLE PRECISION	
datetime	DATE (YYYY-MM-DD)	ViewDate DATE
	TIME (HH:MM:SS)	ViewTime TIME
large object	CHARACTER LARGE OBJECT	
	BINARY LARGE OBJECT	

Scalar Operators

Data Type	Meaning	Example
CHAR_LENGTH	length of string in characters	CHAR_LENGTH(‘Bee’)
LOWER	convert letters to lower-case	LOWER(name)
UPPER	convert letters to upper-case	UPPER(name)
SUBSTRING	returns a substring	SUBSTRING(‘Beech’ FROM 1 TO 3)
CURRENT_DATE	returns the current date	
CURRENT_TIME	returns the current time	

Integrity Enhancement Feature (IEF)

Five types of Integrity constraints defined in CREATE & ALTER:

- Required data
- Domain constraints
- Entity integrity
- Referential integrity
- Enterprise constrains

Required Data

- Null is distinct from blank or zero.
- When NOT NULL is specified, the system rejects any attempt to insert a null in the column.
- If NULL is specified, the system accepts NULL.

Syntax:

```
columnName   dataType   [NOT NULL | NULL]
```

Example:

```
position  VARCHAR(10)  NOT NULL
```

Domain Constraints CHECK

Syntax:

```
CHECK (search condition)
```

Example:

```
sex  CHAR  NOT NULL
      CHECK  (sex  In  (‘M’, ‘F’))

salary  DECIMAL  NOT NULL
      CHECK  (salary > 10000);

bno  INT
       CHECK  (  bno  IN  (SELECT  branchno FROM  branch) )
```

Domain Constraints
DOMAIN

Syntax:

```
CREATE  DOMAIN   domainName  [AS]  datatype
      [DEFAULT   default option ]
      [CHECK  (search condition)];
```

<u>Example</u>:

```
CREATE  DOMAIN  SexType  AS  CHAR
 DEFAULT  ‘M’   CHECK  (VALUE  IN  (‘M’, ‘F’));

 CREATE  DOMAIN  BranchNumber  AS  CHAR(4)
 CHECK  (VALUE  IN  (SELECT  bno  FROM  branch));
```

Domain Constraints DOMAIN

Syntax:

```
        DROP  DOMAIN  DomainName  [RESTRICT | CASCADE];
```

- RESTRICT, domain must not be used in any existing table, view or assertion.
- CASCADE, any column based on the domain is automatically changed to use the underlying data type, column constraint and default clause.

Entity Integrity
PRIMARY KEY

Syntax:

```
PRIMARY KEY (attribute (,…))
```

Example:

PRIMARY KEY (pno)

PRIMARY KEY (cno, pno)

- SQL rejects any operations that attempt to create duplication in the PK column.
- PK forbids NULL value.

Entity Integrity
UNIQUE

- UNIQUE permits NULL value.
- Every column that appears in a UNIQUE clause must also be declared in as NOT NULL.
- UNIQUE can appear after a column definition or separately.

Syntax:

- `UNIQUE(attribute  (,…))`
- `columnName   dataType   [NOT NULL| NULL]  [UNIQUE]`

Example:

```
cno   VARCHAR(5)   NOT   NULL;
pno   VARCHAR(5)   NOT   NULL;
UNIQUE(cno, pno);
pno   VARCHAR(5)   NOT   NULL   UNIQUE;
```

Referential Integrity
FOREIGN KEY

FOREIGN KEY clause is defined in the CREATE & ALTER TABLE statements.

Syntax:

```
        FOREIGN KEY (FK column (,…))
            REFERENCES   table_name [(CK column (,…))]
```

Example:

```
         FOREIGN  KEY (bno) REFERENCES  branch ;
         FOREIGN  KEY (bno) REFERENCES  branch (branchNo);
```

- SQL rejects any INSET or UPDATE operation that attempts to create a foreign key value without a matching CK value key.
- UPDATE or DELETE operation for a CK clause that has matching rows in another table is dependent on the referential action specified using ON UPDATE & ON DELETE subclauses.

Referential Integrity

Four options are supported when the user attempt to delete or update a CK, & there are matching FKs:

- **CASCADE:** automatically delete/update the CK row & all matching (FKs) rows in child table.
- **SET NULL:** delete/update the CK row & set the FK values to NULL. Valid only if NOT NULL clause is not specified for the FK.
- **SET DEFAULT:** delete/update the CK row & set the FK values to default. Valid only if DEFAULT clause is specified for the FK.
- **NO ACTION:** rejects the delete/update operation.

Syntax:

```
 FOREIGN  KEY  ( FK column  (,…) )
   REFERENCES tablename [  ( CK column (,…) )  ]
   [ ON  UPDATE  [ CASCADE | SET NULL| SET DEFAULT| NO ACTION ]  ]
   [ ON  DELETE  [ CASCADE | SET NULL| SET DEFAULT| NO ACTION ]  ]
```

Referential Integrity

<u>Example</u>:

```
FOREIGN KEY  (staffNo)   REFERENCES   staff   ON DELETE  SET  NULL;

FOREIGN KEY  (ownerNo)  REFERENCES  owner  ON UPDATE  CASCADE;

FOREIGN KEY  (MSSN)  REFERENCES  employee (SSN)
        ON DELETE  SET DEFAULT   ON UPDATE  CASCADE;
```

Naming Constraints

In order to modify or delete an existing constraint, it is necessary that the constraint have a name.

Proceed the constraint by the CONSTRIANT clause then specify a name for the constraint.

Example:

```
Sex   CHAR  CONSTRAINT  SexTypeValid
                        CHECK  (sex  IN (‘F’, ‘M’) )

Dept   CHAR(4)   NOT NULL CONSTRAINT  DepNoInList
    CHECK( Dno IN (SELECT Dept  FROM DEPARTMENT))

CONSTRAINT IDISKey
             PRIMARY KEY (SSN)
```

Creating a DB

Syntax

```
CREATE  DATABASE  database_name;
```

Example:

CRETAE DATABSE company;

Dropping a DB

Syntax

```
DROP  DATABASE database_name;
```

Example

```
DROP DATABSE company;
```

Notations

Notations to define SQL statements:

- UPPER-CASE letters represents reserved words.
- Lower-case letters represents user-defined words.
- | indicates a choice among alternatives; (e.g. a | b | c).
- { } indicates a **required** element.
- [ ] indicates an **optional** element.
- … indicates **optional** repetition of an item zero or more times.
- <u>Underlined</u> words represent default values.

Creating a Table

Syntax

```
CREATE  TABLE  tablename
  {  ( {columnName   dataType    [NOT NULL | NULL]   [UNIQUE]
          [DEFAULT   defaultOption ]
          [CHECK     (search condition)] (,…)   }
        [PRIMARY KEY   (column (,…)   ) ,     ]
        [UNIQUE        (column (,…)   ) (,…) ]
        [FOREIGN  KEY  (FK column(,…))
         REFERENCES tablename [(CK column(,…))]
         [ON  UPDATE  ReferentialAction]
         [ON  DELETE  ReferentialAction] (,…) ]
        [CHECK  (search condition)        (,…) ] ) } ;
```

- DEFAULT clause provide a default value for a particular column.
- PRIMARY KEY clause specify the column(s) that form the table’s PK.
- FOREIGN KEY clause specify a foreign key in the table and relate it to another table.
- Column-Based CHECK vs. Tuple-Based CHECK.
- Constraints may be given names using CONSTRAINT clause.

Creating a Table

DEPARTMENT(Dname, Dnumber)

```
CREATE TABLE  department (
        Dname       VARCHAR(15)   NOT NULL,
        Dnumber     INT           NOT NULL,
        PRIMARY KEY (Dnumber),
        UNIQUE (Dname),
        CHECK  (Dname  NOT LIKE  ‘% Inc.’  AND  Dnumber > 70)
);
```

Creating a Table

EMPLOYEE(Fname, Lname, SSN, DOB, Address, Sex, Salary, Dno)

```
CREATE DOMAIN SexType AS  CHAR
  CHECK  (VALUE  IN (‘M’, ‘F’));

CREATE TABLE  employee (
        Fname           VARCHAR(15)   NOT NULL,
        Lname           VARCHAR(15)   NOT NULL,
        SSN             CHAR(9)       NOT NULL,
        DOB             DATE,
        Address         VARCHAR(30),
        Sex             SexType       DEFAULT  ‘F’,
        Salary          DECIMAL(10,2),
        Dno             INT           NOT NULL,
        PRIMARY KEY (SSN),
        FOREIGN KEY (Dno)  REFERENCES  DEPARTMENT(Dnumber)
         ON DELETE  SET DEFAULT  ON UPDATE  CASCADE
 );
```

Creating a Table

```
CREATE TABLE  employee   (
      …… ,
      Sex            CHAR        DEFAULT  ‘F’
       CONSTRAINT    SexValue  CHECK  (Sex IN (‘M’, ‘F’)) ,

      CONSTRAINT   EmpSSN  PRIMARY KEY (SSN),

      CONSTRAINT   EmpFK
      FOREIGN KEY (Dno)  REFERENCES DEPARTMENT(Dnumber)
       ON DELETE  SET DEFAULT  ON UPDATE  CASCADE
);
```

Changing a Table Definition

ALTER consists of six options to:

- Add a column to table
- Drop a column from a table
- Add a table constraint
- Drop a table constraint
- Set a default for a column
- Drop a default for a column

Changing a Table Definition

Syntax

```
ALTER  TABLE  tablename
   [ADD [COLUMN]  ColumnName   dataType  [NOT NULL]  [UNIQUE]
          [DEFAULT  defaultOption]  [CHECK  (search condition)] ]
   [DROP[COLUMN]  ColumnName  [RESTRICT  | CASCADE]]
   [ADD [CONSTRAINT  [Constraint Name]]  TableConstraint Definition]
   [DROP CONSTRAINT  ConstraintName  [RESTRICT | CASCADE]]
   [ALTER  ColumnName  SET DEFAULT  DefaultOption]
   [ALTER  ColumnName  DROP DEFAULT] ;
```

- RESTRICT, drop operation is rejected if the column is referenced by another database object.
- CASCADE, drop operation drops all column from objects it is referenced by.

Changing a Table Definition

Example:

Add an attribute for keeping track of jobs of staff in the company schema.

```
ALTER  TABLE  company.staff
  ADD  job  VARCHAR(12);
```

Example:

Remove the address attribute from the staff table.

```
ALTER  TABLE  company.staff
  DROP   address CASCASE;
```

Changing a Table Definition

<u>Example</u>:

Change the staff table by removing the default of ‘Assistant’ for the position column and setting the default for the sex column to female.

```
ALTER  TABLE  staff
  ALTER  position  DROP DEFAULT;

ALTER  TABLE  staff
  ALTER  sex  SET DEFAULT  ‘F’;
```

Changing a Table Definition

Example:

Change the PropertyForRent table by removing the constraint that the staff are not allowed more than 100 properties at a time (`StaffNotHandlingTooMuch`).

```
ALTER  TABLE  PropertyForRent
  DROP CONSTRAINT StaffNotHandlingTooMuch CASCADE;
```

Example:

Change the employee table by making name a primary key other than Id.

```
ALTER  TABLE  Employee
  DROP CONSTRAINT IDISKey CASCADE
  ADD  CONSTRAINT NameIsKey PRIMARY KEY (name);
```

Changing a Table Definition

Example:

Change the Client table by adding a new column representing the preferred number of rooms.

```
ALTER  TABLE  Client
  ADD PrefNoRooms PropertyRooms;
```

Removing a Table

Syntax

```
   DROP  TABLE  tablename  [RESTRICT | CASCADE];
```

- RESTRICT, drop operation is rejected if there are any other objects that depend for their existence upon the existence of the table to be dropped.

- CASCADE, drop operation drops all dependent objects.

Creating an Index

Index is a structure that provides accelerated access to rows of a table based on the value of one or more attributes.

Indexes are updated every time the underlying tables are modified.

Created only on base tables.

Syntax

```
CREATE  [UNIQUE] INDEX  IndexName
 ON  tableName  (columnName  [ASC | DESC] [,…])
```

Example:

CRETAE UNIQUE INDEX StaffInd ON staff (StaffNo);

CREATE INDEX RentInd ON PropertyForRent (city, rent);

Removing an Index

Syntax

```
DROP   INDEX   Indexname;
```

SQL-DML

Data Manipulation Language (DML) Statements

The main SQL data manipulation language statements are:

SELECT

INSERT INTO

UPDATE

DELETE FROM

Simple Queries

Syntax

```
SELECT [DISTINCT|ALL]{*|column|column_expression [AS new_name][,…]}
   FROM   table_name [alias] [, … ]
     [WHERE  condition]
     [GROUP BY  column_list]
     [HAVING  condition]
     [ORDER BY  column_list [ASC|DESC]];
```

- *column* represents a column name.
- *column_expression* represents an expression on a column.
- *table_name* is the name of an existing database table or view.
- FROM specifies the table(s) to be used.
- WHERE filters the rows subject to some condition.
- GROUP BY forms groups of rows with the same column name.
- SELECT specifies which column are to appear in the output.
- ORDER BY specifies the order of the output.
- Order of the clauses in the SELECT statement can not be changed.
- The result of a query is another table.
- Asterisk (*) means all columns.

Simple Queries
Retrieve all columns & rows

Syntax

```
SELECT {* | column| column_expression [,…]}
   FROM   table_name;
```

<u>Example</u>: STAFF(sno, fname, lname, position, sex, dob, salary, bno)

Retrieve all staff information.

```
SELECT  sno, fname, lname, position, sex, dob, salary, bno
  FROM  staff;
```

OR

```
SELECT  *
  FROM  staff;
```

Simple Queries
Retrieve specific columns & all rows

Example: STAFF(sno, fname, lname, position, sex, dob, salary, bno)

List salaries of all staff, showing only the staff number, the first and last name, and salary.

```
SELECT  sno, fname, lname, salary
  FROM  staff;
```

Simple Queries
Use of DISTINCT

DISTINCT eliminates duplicated tuples.

Syntax

```
SELECT [DISTINCT|ALL] {* | column |column_expression [,…]}
   FROM   table_name;
```

Example: STAFF(sno, fname, lname, position, sex, dob, salary, bno)

List the available positions for staff .

```
SELECT DISTINCT position
 FROM  staff;
```

position
Manager
Assistant
Supervisor
Assistant
Manager

```
SELECT position
 FROM  staff;
```

position
Manager
Assistant
Supervisor

```
SELECT DISTINCT position
 FROM  staff;
```

Simple Queries
Calculated fields

- The SQL expression in the SELECT list specifies a derived field.
- Columns referenced in the arithmetic expression must have a numeric type.
- SQL expression can involve + , - , * , / , (,).
- AS clause is used to name the derived column.

Syntax

```
SELECT {* | column| column_expression [AS new_name] [,…]}
   FROM   table_name;
```

<u>Example</u>: STAFF(sno, fname, lname, position, sex, dob, salary, bno)

List the monthly salaries for all staff, showing the staff number, the first and last names.

```
SELECT sno, fname, lname, salary/12 AS MonthlySalary
 FROM  staff;
```

Simple Queries
Row selection (WHERE clause)

WHERE clause consists of five basic search conditions:

- **Comparison:** Compare the value of one expression to the value of another expression (= , <, >, <=, >=, <>).
- **Range:** Test whether the value of an expression falls within a specified range of values (BETWEEN/ NOT BETWEEN).
- **Set membership:** Test whether the value of an expression equals one of a set of values (IN/ NOT IN).
- **Pattern match:** Test whether a string matches a specified pattern (LIKE/ NOT LIKE).
- **NULL:** Test whether a column has null value (IS NULL/ IS NOT NULL).

Simple Queries
Comparison search condition

Comparison operators: = , < , > , <= , >= , <>

Syntax

```
SELECT [DISTINCT|ALL] {* | column| [column_expression [AS
new_name]] [,…]}
   FROM   table_name
     [WHERE  condition];
```

Example: STAFF(sno, fname, lname, position, sex, dob, salary, bno)

List all staff with a salary greater than 10,000. showing number, name and salary.

```
SELECT sno, fname, lname, salary
  FROM staff
    WHERE  salary > 10000;
```

Simple Queries
Compound comparison search condition

Compound comparison operators: AND , OR , NOT , ( )

Order of evaluation:

- Expression is evaluated left to right
- Between brackets
- NOT
- AND
- OR

Example: STAFF(sno, fname, lname, position, sex, dob, salary, bno)

List all staff who works as managers or assistants.

```
SELECT sno, fname, lname, position
 FROM staff
  WHERE  position = ‘Manager’ OR position = ‘Assistant’;
```

Simple Queries
BETWEEN/ NOT BETWEEN

BETWEEN checks if a value is within a range.
NOT BETWEEN checks if a value is outside a range.

Example: STAFF(sno, fname, lname, position, sex, dob, salary, bno)

List all staff with a salary between 20000 and 30000.

```
SELECT sno, fname, lname, salary
  FROM staff
    WHERE  salary BETWEEN 20000 AND 30000;
```

This would be expressed as:

```
SELECT sno, fname, lname, salary
  FROM staff
    WHERE  salary >= 20000 AND salary <= 30000;
```

Simple Queries
IN/ NOT IN

IN tests whether a data value matches one of a list values.
NOT IN checks for data values that do not lie in a specific list of values.

Example: STAFF(sno, fname, lname, position, sex, dob, salary, bno)

List all Managers or Assistants.

```
SELECT sno, fname, lname, position
   FROM  staff
      WHERE   position IN (‘Manager’, ‘Assistant’);
```

This would be expressed as:

```
SELECT sno, fname, lname, position
  FROM  staff
   WHERE   position = ‘Manager’ OR position = ’Assistant’;
```

Simple Queries
LIKE/ NOT LIKE

SQL has special pattern matching symbol:

% represents any sequence of zero or more character (wildcard)

_ represents any single character

Example:

- Address LIKE ‘H%’ means that the first character must be *H*, but the rest can be anything.
- Address LIKE ‘H_ _ _’ means that there must be exactly four characters in the string, the first of which must be *H.*
- Address LIKE ‘%e’ means any sequence of characters, of length at least 1, with the last character an *e.*
- Address LIKE ‘%Glasgow%’ means a sequence of characters of any length containing *Glasgow.*
- Address NOT LIKE ‘H%’ means the first character can not be *H.*

Simple Queries
LIKE/ NOT LIKE

If the search string can include the pattern-matching character itself, we can use an **escape** character to represent the pattern matching character.

‘15%’ is represented by LIKE ‘15#%’ ESCAPE ‘#’

Example: STAFF(sno, fname, lname, position, sex, dob, salary, address, bno)

List all staff with the string ‘Glasgow’ in their address.

```
SELECT sno, fname, lname, address
   FROM  staff
      WHERE   address LIKE ‘%Glasgow%’;
```

Simple Queries
IS NULL/ IS NOT NULL

NULL represents missing or unknown value.

NULL can does not represent a zero or a string of blank spaces.

A NULL value can not be tested with = or<> to another string.

We have to test for NULL explicitly.

Example:

VIEWING (ClientNo, PropertyNo, ViewDate, Comment)

List the details of all viewing on property PG4 where a comment has not been supplied.

```
SELECT clientno, ViewDate
   FROM  viewing
     WHERE  PropertyNo= ‘PG4’ AND  comment IS NULL;
```

Question

Assume the following relational schema:

EMPLOYEE(Fname, Lname, SSN, DOB, Address, Sex, salary, DeptNo)
DEPARTMENT(Dname, DNo)
PROJECT(PName, PNo, PLocation, Dno)
WORKS_ON(SSN, PNo, Hours)

List all employees in department 5 whose salary is between Birr 30,000 & Birr40,000.

Simple Queries
ORDER BY clause

Allows the retrieved records to be ordered in ascending (ASC) or descending order (DESC) on any column or combination of columns.

Syntax

```
SELECT {* | [column_expression] [,…]}
   FROM   table_name
     [ORDER BY  column_list [ASC|DESC] ]
```

Single Column ordering

STAFF(sno, fname, lname, position, sex, dob, salary, bno)

Produce a list of salaries for all staff, arranged in descending order of salary.

```
SELECT  sno, fname, lname, salary
  FROM  staff
    ORDER BY salary  DESC;
```

Simple Queries
ORDER BY clause

Multiple columns ordering

Property (PropertyNo, Street, City, postcode, Type, OwnerNo, Rooms, Rent)

Produce a list of properties arranged in order of property type and within each property type ordered by rent in descending order.

```
SELECT propertyNo, type, rooms, rent
  FROM property
     ORDER BY  type, rent  DESC;
```

PropertNo	Type	Rooms	Rent
PG16	Flat	4	450
PL94	Flat	4	400
PG36	Flat	3	370
PG4	House	3	650
PA14	House	6	600

Question

Assume the following relational schema:

EMPLOYEE(Fname, Lname, SSN, DOB, Address, Sex, salary, DeptNo)
DEPARTMENT(Dname, DNo)
PROJECT(PName, PNo, PLocation, Dno)
WORKS_ON(SSN, PNo, Hours)

List all employees, ordered by department and, within each department, ordered alphabetically by last name, first name.

Simple Queries
Aggregation

Functions that operate on a single column of a table and return a single value.

Five aggregation functions defined in SQL:

COUNT returns the number of rows in a specified column.
SUM returns the sum of the values in a specified column.
AVG returns the average of the values in a specified column.
MIN returns the smallest value in a specified column.
MAX returns the largest value in a specified column.

Examples:

Property (PropertyNo, Street, City, postcode, Type, OwnerNo, Rooms, Rent)

How many properties cost more than 350 per month to rent?

```
SELECT COUNT(*) AS count
   FROM  property
     WHERE   rent > 350;
```

count
2

Simple Queries
Aggregation

VIEWING (ClientNo, PropertyNo, ViewDate, Comment)

How many different properties were viewed in May 1998?

```
SELECT  COUNT(DISTINCT  PropertyNo)  AS  count
  FROM  viewing
    WHERE  Viewdate   BETWEEN  ‘1-May-98’  AND  ‘31-May-98’;
```

count
2

Simple Queries
Aggregation

STAFF(sno, fname, lname, position, sex, dob, salary, bno)

Find the total number of Managers and the sum of their salaries.

```
SELECT  COUNT(sno)  AS  count,  SUM(salary)  AS  sum
   FROM   staff
     WHERE   position  =  'Manager';
```

count	sum
2	54000

Simple Queries Aggregation

STAFF(sno, fname, lname, position, sex, dob, salary, bno)

Find the minimum, maximum, and average staff salary.

```
SELECT MIN(salary) AS  min,  MAX(salary)  AS  max,
AVG(salary)  AS avg
  FROM staff;
```

min	max	avg
9000	30000	17000

Simple Queries
GROUP BY clause

Groups the data from the SELECT table(s) and produces a single summary row for each group.

Example:

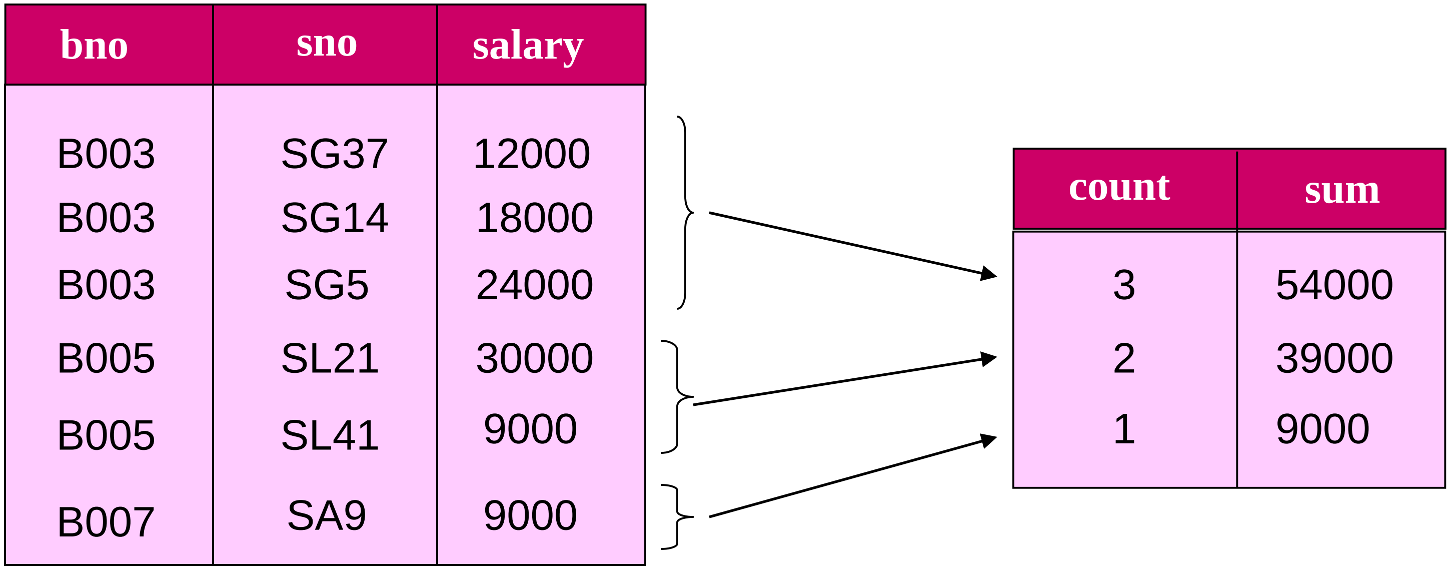
STAFF(sno, fname, lname, position, sex, dob, salary, bno)

Find the number of staff working in each branch and the sum of their salaries.

```
SELECT  bno, COUNT(sno) AS count, SUM(salary) AS sum
  FROM  staff
    GROUP BY bno;
```

bno	count	sum
B003	3	54000
B005	2	39000
B007	1	9000

Simple Queries
GROUP BY clause



bno	sno	salary
B003	SG37	12000
B003	SG14	18000
B003	SG5	24000
B005	SL21	30000
B005	SL41	9000
B007	SA9	9000

count	sum
3	54000
2	39000
1	9000

Simple Queries
HAVING clause

Designed for use with the GROUP BY clause to restrict the groups that appear in the final result table.
WHERE clause filters individual rows going into the final result table.
HAVING clause filters groups going into the final result table.

<u>Example</u>:

STAFF(sno, fname, lname, position, sex, dob, salary, bno)

For each branch office with more than one member of staff, find the number of staff working in each branch and the sum of their salaries.

```
SELECT  bno, COUNT(sno) AS count, SUM(salary) AS sum
  FROM  staff
    GROUP BY bno
      HAVING COUNT(sno) > 1;
```

bno	count	sum
B003	3	54000
B005	2	39000

Question

Assume the following relational schema:

EMPLOYEE(Fname, Lname, SSN, DOB, Address, Sex, salary, DeptNo)
DEPARTMENT(Dname, DNo)
PROJECT(PName, PNo, PLocation, Dno)
WORKS_ON(SSN, PNo, Hours)

For each project on which more than two employees work, retrieve the project number and the number of employees who work on the project.

Subqueries

A complete SELECT statement can be embedded (subselect) within another SELECT statement.

A subselect can be used in the WHERE and HAVING clauses of the outer SELECT statement (nested query).

A subquery can be used immediately following a relational operator.

Subquery always enclosed in parentheses.

Type of subquery:

- A *scalar subquery* returns a single column and a single row (singlevalue).
- A *row subquery* returns multiple columns, but a single row.
- A *table subquery* returns one or more columns and multiple rows.

Subqueries

STAFF (sno, fname, lname, position, sex, DOB, salary, bno)
BRANCH (bno, street, city, postcode)

Example:
List the staff who work in the branch at ‘163 Main St’.

```
SELECT  sno,  fname,  lname,  position
  FROM staff
    WHERE  bno  = (SELECT  bno
                    FROM  branch
                     WHERE  street = ‘163 Main St’);
```

Subqueries

STAFF (sno, fname, lname, position, sex, DOB, salary, bno)

Example:

List the staff whose salary is greater than the average salary, and list by how much their salary is greater than the average.

```
SELECT sno, fname, lname, position, salary – (SELECT
       avg(salary) FROM staff ) AS sal_diff
  FROM staff
    WHERE  salary  > ( SELECT  avg(salary)
                        FROM  staff  );
```

Subqueries

The following rules apply to subqueries:

- The ORDER BY clause may not be used in a subquery .
- The subquery SELECT list must consist of a single column name or expression, except for subqueries that use the keyword EXISTS.
- By default, column names in a subquery refer to the table name in the FROM clause of the subquery. It is possible to refer to a table in a FROM clause in an outer query by qualifying the column name; in this case the subquery is called a *correlated subquery*.
- When a subquery is one of the two operands involved in a comparison, the subquery must appear on the right-hand side of the comparison.

Subqueries
IN

PROPERTYFORRENT (pno, street, area, city, pcode, type, rooms, rent, sno)
STAFF (sno, fname, lname, position, sex, DOB, salary, bno)
BRANCH (bno, street, city, postcode)

Example:
List the properties that are handled by staff who work in the branch at ‘163 Main St’.

```
SELECT pno, street, area, city, pcode, type, rooms, rent
 FROM property_for_rent
   WHERE sno  IN
       (SELECT  sno
          FROM  staff
            WHERE  bno =
                    (SELECT  bno
                      FROM  branch
                         WHERE  street  =  ‘163 MainSt’));
```

Question

Assume the following relational schema:

EMPLOYEE(Fname, Lname, SSN, DOB, Address, Sex, salary, DeptNo)
DEPARTMENT(Dname, DNo)
PROJECT(PName, PNo, PLocation, Dno)
WORKS_ON(SSN, PNo, Hours)

Show the resulting salaries if every employee working on ‘X’ project is given all %10 raise.

Subqueries
ANY/ ALL

- Used with subqueries that produce a single column of numbers.
- If the subquery is preceded by the keyword ALL, the condition will only be true if it is satisfied by all values produced by the subquery.
- If the subquery is preceded by the keyword ANY or SOME, the condition will be true if it is satisfied by any (one or more) values produced by the subquery.

Subqueries
ANY/ ALL

STAFF (sno, fname, lname, position, sex, DOB, salary, bno)

Example:

Find staff whose salary is larger than the salary of at least one member of staff at branch B3.

```
SELECT  sno,  fname,  lname,  position, salary
  FROM staff
    WHERE salary > SOME
           (SELECT salary
             FROM  staff
              WHERE bno = ‘B3’);
```

Subqueries
ANY/ ALL

STAFF (sno, fname, lname, position, sex, DOB, salary, bno)

Example:

Find staff whose salary is larger than the salary of every member of staff at branch B3.

```
SELECT  sno,  fname,  lname,  position, salary
  FROM staff
    WHERE salary > ALL
           (SELECT salary
             FROM  staff
              WHERE bno = ‘B3’);
```

Homework Question

Assume the following relational schema:

EMPLOYEE (Fname, Lname, SSN, DOB, Address, Sex, salary, DeptNo)
DEPARTMENT (Dname, DNo)
PROJECT (PName, PNo, PLocation, Dno)
WORKS_ON(SSN, PNo, Hours)

For each department that has more than 5 employees, retrieve the department number and the number of its employees who are making more than $40,000.

Multi-Table Queries

- So far, the columns that are to appear in the result table must all come from a single table.
- To combine columns from several tables into a result table, we need to use a join operation.
- To perform a join, we include more than one table name in the FROM clause. WHERE clause to specify the join columns.

```
SELECT [DISTINCT|ALL] {* |column |[column_expression
                                   [AS new_name]] [,…]}
   FROM   table_name [alias] [, … ]
     [WHERE  condition];
```

Simple Join

CLIENT (ClientNo, Fname, Lname, telNo, Type, Rent)
VIEWING (ClientNo, PropertyNo, Date, Comment)

Example:
List the names of all clients who have viewed a property along with any comment supplied.

```
SELECT  c.clientNo, fname,  lname,  propertyNo,  comment
  FROM  client  c,  viewing  v
    WHERE  c.clientNo  =  v.clientNo;
```

Sorting a Join

PROPERTYFORRENT (pno, street, area, city, pcode, type, rooms, rent, sno)
STAFF (sno, fname, lname, position, sex, DOB, salary, bno)
BRANCH (bno, street, city, postcode)

<u>Example</u>:
For each branch office, list the names of staff who manage properties, and the properties they manage, ordered by branch number, staff number and property number.

```
SELECT  s.bno, s.sno, fname,  lname,  pno
  FROM  staff s,  propertyforrent p
    WHERE  s.sno  =  p.sno
     ORDER BY  s.bno, s.sno, p.pno;
```

Question

Assume the following relational schema:

EMPLOYEE (Fname, Lname, SSN, DOB, Address, Sex, salary, DeptNo)
DEPARTMENT (Dname, DNo)
PROJECT (PName, PNo, PLocation, Dno)
WORKS_ON(SSN, PNo, Hours)

List all employees and identify the projects they are working on, ordered by department and, within each department, ordered alphabetically by last name, first name.

Three-Table Join

PROPERTYFORRENT (pno, street, area, city, pcode, type, rooms, rent, sno)
STAFF (sno, fname, lname, position, sex, DOB, salary,_bno)
BRANCH (bno, street, city, postcode)

Example:
For each branch, list the staff who manage properties, including the city in which the branch is located and the properties they manage.

```
SELECT  b.bno, b.city, s.sno, fname, lname, pno
 FROM  branch b, staff s, propertyForRent p
  WHERE  b.bno = s.bno  AND  s.sno = p.sno;
```

Alternatives:

```
FROM  (Branch b  JOIN  staff s USING  bno)  As  bs
                 JOIN  PropertyForRent p  USING  sno;
```

Multiple grouping columns

PROPERTYFORRENT (pno, street, area, city, pcode, type, rooms, rent, sno)
STAFF (sno, fname, lname, position, sex, DOB, salary, bno)
BRANCH (bno, street, city, postcode)

Exmaple:
Find the number of properties handled by each staff member and branch.

```
SELECT  s.bno,  s.sno,  COUNT(*)  AS  count
  FROM  staff  s,  propertyForRent  p
    WHERE  s.sno = p.sno
       GROUP BY s.bno, s.sno;
```

Computing a Join

A join is a subset of the Cartesian product.

The Cartesian product of two tables is another table consisting of all possible pairs of rows from the two table.

The columns of the product table are all the columns of the first table followed by all the columns of the second table.

Format of SELECT statement for the Cartesian product:

```
SELECT [DISTICNT | ALL]  {* | column_list }
  FROM  table_name1 CROSS JOIN  table_name2;
```

Computing a Join

The procedure for generating the results of a SELECT with a join are as follows:

- Form the Cartesian product of the tables named in the FROM clause.
- If there is a WHERE clause, apply the search condition to each row of the product table, retaining those rows that satisfy the condition. In terms of the relational algebra, this operation yields a restriction of the Cartesian product.
- For each remaining row, determine the value of each item in the SELECT list to produce a single row in the result table.
- If SELECT DISTINCT has been specified, eliminate any duplicate rows from the result table.
- If there is an ORDER BY clause, sort the result table as required.

Outer Join

The **join** operation combines data from two tables by forming pairs of related rows where the matching columns in each table have the same value. If one row of a table is unmatched, the row is omitted from the result table.

Outer join include the unmatched rows in the result table.

Three types of outer join:

- Left
- Right
- Full

Join Example

BRANCH

BranchNo	bCity
B003	Glasgow
B004	Bristol
B002	London

PROPERTY

PropertyNo	pCity
PA14	Aberdeen
PL94	London
PG4	Glasgow

BranchNo	bCity	PropertyNo	pCity
B003	Glasgow	PG4	Glasgow
B002	London	PL94	London

```
SELECT   b.*, p.*
  FROM  branch  b, property p
    WHERE  b.bcity =  p.pcity;
```

Left Outer Join

Example:

List the branch offices and properties that are in the same city along with any unmatched branches.

```
SELECT  b.*, p.*
 FROM  branch  b
    LEFT  JOIN  property  p ON
          b.bcity  =  p.pcity;
```

BRANCH

BranchNo	bCity
B003	Glasgow
B004	Bristol
B002	London

PROPERTY

PropertyNo	pCity
PA14	Aberdeen
PL94	London
PG4	Glasgow

BranchNo	bCity	PropertyNo	pCity
B003	Glasgow	PG4	Glasgow
B004	Bristol	NULL	NULL
B002	London	PL94	London

```
SELECT  b.*, p.*
 FROM  branch  b
  LEFT  JOIN  property  p ON
              b.bcity  =  p.pcity;
```

Right Outer Join

Example:

List the branch offices and properties in the same city and any unmatched property.

```
SELECT  b.*, p.*
  FROM  branch  b
     RIGHT  JOIN  property  p ON
             b.bcity  =  p.pcity;
```

BRANCH

BranchNo	bCity
B003	Glasgow
B004	Bristol
B002	London

PROPERTY

PropertyNo	pCity
PA14	Aberdeen
PL94	London
PG4	Glasgow

BranchNo	bCity	PropertyNo	pCity
NULL	NULL	PA14	Aberdeen
B003	Glasgow	PL94	London
B002	London	PG4	Glasgow

```
SELECT  b.*, p.*
  FROM  branch  b
     RIGHT  JOIN  property  p ON
             b.bcity  =  p.pcity;
```

Full Outer Join

Example:

List the branch offices and properties that are in the same city and any unmatched branches or properties.

```
SELECT  b.*, p.*
  FROM  branch  b
     FULL  JOIN  property  p ON
           b.bcity  =  p.pcity;
```

BRANCH

BranchNo	bCity
B003	Glasgow
B004	Bristol
B002	London

PROPERTY

PropertyNo	pCity
PA14	Aberdeen
PL94	London
PG4	Glasgow

BranchNo	bCity	PropertyNo	pCity
NULL	NULL	PA14	Aberdeen
B003	Glasgow	PG4	Glasgow
B004	Bristol	NULL	NULL
B002	London	PL94	London

```
SELECT  b.*, p.*
  FROM  branch  b
     FULL  JOIN  property  p ON
             b.bcity  =  p.pcity;
```

EXIST/ NOT EXIST

Used only with correlated subqueries.
EXISTS is true if and only if there exists at least one row in the result table returned by the subquery. It is false if the subquery returns an empty result table.

Example:

STAFF (sno, fname, lname, position, sex, DOB, salary, bno)
BRANCH (bno, street, city, postcode)

Find all staff who work in a London branch.

```
SELECT  sno,  fname,  lname,  position
  FROM  staff  s
   WHERE  EXISTS
     (SELECT  *
       FROM  branch  b
         WHERE  s.bno  = b.bno  AND  city  = ‘London’);
```

Question

Assume the following relational schema:

EMPLOYEE (Fname, Lname, SSN, DOB, Address, Sex, salary, DeptNo)
DEPARTMENT (Dname, DNo)
PROJECT (PName, PNo, PLocation, Dno)
WORKS_ON(SSN, PNo, Hours)

Retrieve the names of employees who works on no project.

UNION

PROPERTYFORRENT (pno, street, area, city, pcode, type, rooms, rent, sno)
STAFF (sno, fname, lname, position, sex, DOB, salary, bno)
BRANCH (bno, street, city, postcode)

Example:
Construct a list of all cities where there is either a branch office or a rental property.

```
(SELECT  city   FROM  branch)
UNION
(SELECT  city   FROM  propertyforrent);
```

INTERSECT

Example:

Construct a list of all cities where there is both a branch office and a rental property.

```
(SELECT  city   FROM  branch)
INTERSECT
(SELECT  city   FROM propertyforrent);
```

```
SELECT DISTINCT b.city
  FROM  branch  b, propertyforrent  p
    WHERE  b.city=p.city;
```

```
SELECT DISTINCT city
  FROM   branch  b
   WHERE  EXISTS
           (SELECT  *
                 FROM propertyforrent  p
                    WHERE  p.city  =  b.city);
```

EXCEPT

Example:

Construct a list of all cities where there is a branch office but no rental property.

```
(SELECT  city  FROM  branch)
EXCEPT
(SELECT  city  FROM propertyforrent);
```

```
SELECT  DISTINCT city
  FROM  branch
   WHERE  city NOT IN
          (SELECT   city
            FROM propertyforrent);
```

```
SELECT DISTINCT city
  FROM   branch  b
   WHERE  NOT EXISTS
           (SELECT  *  FROM propertyforrent  p
                    WHERE  p.city  =  b.city);
```

Adding Data to DB (INSERT)

Syntax

```
    INSERT INTO table_name [(column (,…))]
          { VALUES (date_value (,…)) |  subquery };
```

- *table_name* may be either a base table or an updatable view.
- *column_list* represents a list of one or more column names separated by commas.
- If omitted, SQL assumes a list of all columns in their original CREATE TABLE order.
- If specified, then any columns that are omitted from the list must have been declared as NULL column.
- <u>*data_value* must match the *column_list* as follows</u>:
  - The number of items in each list must be same.
  - There must be a direct correspondence in the position of items in the two lists, so that the first item in the *data_value_list* applies to the first item in the *column_list*, and so on.
  - The data type of each item in the *data_value_list* must be compatible with the data type of the corresponding column.

Simple INSERT

STAFF(sno, fname, lname, position, sex, DOB, salary, bno)

Example:
Insert a new row into the staff table supplying data for all columns.

```
INSERT INTO staff
VALUES (‘SG16’, ‘Alan’, ‘Brown’, ‘Assistant’, ‘M’,
         DATE ‘1957-05-25’, 8300, ‘B003’);
```

Simple INSERT

STAFF(sno, fname, lname, position, sex, DOB, salary, bno)

<u>Example</u>:
Insert a new row into the staff table supplying data for all mandatory columns, knowing that the sex and birth date are optional fields.

```
INSERT INTO staff (Sno, fname, lname, position, salary, bno)
VALUES (‘SG16’, ‘Alan’, ‘Brown’, ‘Assistant’, 8300, ‘B003’);
```

Alternative:

```
INSERT INTO staff
VALUES (‘SG16’, ‘Alan’, ‘Brown’, ‘Assistant’, NULL, NULL, 8300,
       ‘B003’);
```

INSERT with subqueries

STAFF(sno, fname, lname, position, sex, DOB, salary, bno)
PROPERTYFORRENT(Pno, street, city, postcode, type, rooms, rent, ono, sno, bno)
StaffPropCount(sno, fname, lname, propcount)

Example:
Insert rows into the StaffPropCount table using the staff and property_for_rent tables.

```
INSERT INTO staffPropCount
(SELECT  s.sno, fname, lname, COUNT(*)
  FROM  staff s, PropertyForRent p
   WHERE  s.sno = p.sno
    GROUP BY s.sno, fname, lname)
UNION
(SELECT  sno, fname, lname, 0
  FROM  Staff
    WHERE  sno NOT IN (SELECT DISTINCT sno
                           FROM  PropertyForRent));
```

Modifying Data in the DB (UPDATE)

Syntax

```
    UPDATE table_name
       SET column_name1 = data_value1 [, column_namei =
data_valuei ...]
            [WHERE  search_condition]
```

- *table_name* may be either a base table or an updatable view.
- The SET clause specifies the names of one or more columns that are updated for all rows in the table.
- Only rows that satisfy the *search_condition* are updated.
- *data_values* must be compatible with the data types for the corresponding columns.

Simple UPDATE

STAFF(sno, fname, lname, position, sex, DOB, salary, bno)

<u>Example</u>:
Give all staff a 3% pay increase.

```
UPDATE  staff
  SET  salary = salary * 1.03;
```

<u>Example</u>:
Give all managers a 3% pay increase.

```
UPDATE  staff
  SET  salary = salary * 1.03
    WHERE  position = ‘Manager’;
```

Simple UPDATE

STAFF(sno, fname, lname, position, sex, DOB, salary, bno)

Example:
Promote David Ford (sno = ‘SG14’) to Manager and change his salary to $18,000.

```
UPDATE  staff
  SET  position=‘Manager’, salary = 18000
   WHERE  sno=‘SG14’;
```

Deleting Data from the DB (DELETE)

Syntax

```
    DELETE  FROM  table_name
                  [WHERE  search_condition];
```

- *table_name* may be either a base table or an updatable view.
- Only rows that satisfy the *search_condition* are deleted.
- If no *search_condition* is omitted, all rows are deleted from the table.
- DELETE does not delete the table itself, only rows in the table.

Simple DELETE

STAFF(sno, fname, lname, position, sex, DOB, salary, bno)

Example:
Delete all staff in branch B003.

```
DELETE FROM  staff
  WHERE  bno = ‘B003’;
```

Example:
Delete all staff.

```
DELETE FROM staff;
```

Creating a View

Syntax

```
CREATE  VIEW  ViewName  [(newColumnName  [,…])]
 AS  subselect
   [WITH  [CASCADE | LOCAL]  CHECK  OPTION]
```

NewColumnName assign a name to each column in view.

If WITH CHECK OPTION is specified, if a row fails to satisfy the WHERE clause, it is not added to the base table of the view.

VIEWS

Creating a View (Horizontal)

Example:
Create a view for managers at branch B003 can see only the details for staff who work in their branch office.

```
CREATE VIEW  Maneger3Staff
 AS SELECT  *
   FROM  staff
    WHERE   branchNo = ‘B003’;
```

```
SELECT  *
  FROM   Manager3Staff;
```

Creating a View (Vertical)

Example:
Create a view for staff details at branch B003 that excludes salary information.

```
CREATE VIEW  Staff3
 AS SELECT  StaffNo, Fname, Lname, position, sex
     FROM  staff
       WHERE   branchNo = ‘B003’;
```

```
CREATE VIEW  Staff3
 AS SELECT  StaffNo, Fname, Lname, position, sex
     FROM  Manager3Staff;
```

Creating a View (Groups & Join)

PROPERTYFORRENT (pno, street, area, city,pcode, type, rooms,rent,sno)
STAFF (sno, fname, lname, position, sex, DOB, salary, bno)

Example:

Create a view for staff who manage properties for rent, which include the branch number they work at, their staff number, and the number of properties they manage.

```
CREATE VIEW  StaffPropCount  (BranchNo, StaffNo, cnt)
 AS SELECT  s.Bno,  s.Sno, COUNT(*)
    FROM  Staff  s,  PropertyForRent  p
      WHERE   s.sno = p.sno
        GROUP  BY  s.bno, s.sno;
```

BranchNo	StaffNo	Cnt
B003	SG14	1
B003	SG37	2
B005	SL41	1
B007	SA9	1

Removing a View

Syntax

```
        DROP  VIEW  ViewName  [RESTRICT | CASCADE];
```

- RESTRICT, drop operation is rejected if there are any other objects that depend for their existence upon the existence of the view to be dropped.

- CASCADE, drop operation drops all views defined on the dropped view.

Restrictions on Accessing Views

1. If a column in the view is based on aggregation function, then the column may appear only in SELECT and ORDER BY clauses of queries that access the view.

Example: Illegal operation

```
SELECT  COUNT(cnt)
    FROM   StaffPropCount;


SELECT  *
    FROM   StaffPropCount
       WHERE  cnt  >  2;
```

2. Grouped view may never be joined with a base table or a view.

View Updatability

For a view to be updatable, the DBMS must be able to trace any row or column back to its row or column in the source table.

All updates to a base relation are immediately reflected in all views that reference that base relation.

All updates to a view will be reflected in the underlying base relation, under the following rules:

- Updates are allowed through a view defined using a simple query involving a single base relation & containing either the PK or a CK of the base relation.
- Update are NOT allowed through views involving multiple base relations.
- Updates are NOT allowed through views involving aggregation or grouping operations.

View Updatability

<u>Example</u>: Illegal operation

```
INSERT INTO StaffPropCount
  VALUES  (‘B003’,  ‘SG5’,  2);
```

View Updatability

<u>Example</u>: Illegal operation

```
CREATE  TABLE  PropertyForRent(
        Pno     VARCHAR(5)    NOT NULL,
        city    VACHAR(15)    NOT NULL,
         …   );
```

```
CREATE VIEW  StaffPropList (bno, sno, pno)
   AS  SELECT  s.bno,  s.sno,  p.pno
       FROM  staff  s,  propertyforrent p
       WHERE  s.sno = p.sno;
```

```
 INSERT INTO StaffPropList
    VALUES  (‘B003’,  ‘SG5’,  ‘PG19’);
```

View Updatability

A view is updatable if:

- DISTINCT not specified.
- Every column in the SELECT statement is a column name (rather than constant, expression, or aggregation function).
- FROM clause specifies only one table.
- The WHERE clause does not include any subquesries.
- There is no GROUP BY or HAVING clause.
- Any row inserted through the view must not violate the integrity constraints of the base table.

WITH CHECK OPTION

If a row in a view is altered in a way that is no longer satisfies the condition, then it will disappear from the view **(migration rows).**

WITH CHECK OPTION prohibits a row migration out of the view.

- When INSERT or UPDATE statement on the view violates the WHERE condition, the operation is rejected.

[LOCAL | CASCADE] applicable to view hierarchies.

- **LOCAL**, any row inserted or updated on this view, must not cause the row to disappear from the view, unless the row also disappears from the underlying derived view/table.
- **CASCADE**, any row inserted or updated on this view and view defined on this view must not cause the row to disappear from the view.

WITH CHECK OPTION

Example:

```
CREATE VIEW Manager3Staff
     AS   SELECT  *
          FROM   Staff
       WHERE  bno  =  ‘B003’
     WITH  CHECK  OPTION;
```

```
UPDATE  Manager3Staff
    SET   bno  =  ‘B005’
      WHERE  sno = ‘SG37’;
```

```
INSERT  INTO  Manager3Staff
   VALUES  (‘SL15’,  ‘Mary’,  ‘Black’,  ‘Assistant’,  ‘F’,
             DATE  ‘1987-06-12’, 8000, ‘B002’);
```

WITH CHECK OPTION

Example:

```
CREATE VIEW LowSalary
     AS    SELECT   *
           FROM    Staff
        WHERE   salary > 9000;
```

```
CREATE VIEW Manager3Staff
   AS    SELECT   *
     FROM    HighSalary
        WHERE   bno = ‘B003’;
```

```
CREATE VIEW HighSalary
    AS    SELECT   *
          FROM    LowSalary
      WHERE   salary > 10000
WITH   LOCAL   CHECK   OPTION;
```

```
UPDATE   Manager3Staff
    SET   Salary   =   9500
       WHERE    Sno = ‘SG37’;
```

SQL-DCL

Data Control Language

- **Data Control Language** (DCL) statements control the level of access that users have on database objects. They are:
  - GRANT - gives user's access privileges to database
  - REVOKE - withdraw access privileges given with the GRANT command

Privileges

- Allowable Privileges
  - SELECT, INSERT, UPDATE, DELETE
  - CREATE Table, View, Procedure, Trigger, Rule, Default
- The owner/creator of a table automatically has all the privileges

GRANT/ REVOKE

- **GRANT privilege ON tablename TO list**

  [WITH GRANT OPTION]
- For example
  - GRANT ALL ON BRANCH TO Solomon
  - GRANT SELECT ON BRANCH TO sally WITH GRANT OPTION
  - GRANT SELECT, UPDATE, INSERT ON BRANCH TO Solomon, Girma, Senait
- REVOKE privilege ON tablename FROM list [CASCADE]
- Example
  - REVOKE SELECT ON BRANCH FROM Girma CASCADE

Thank for Your Attention!