**Chapter 6: SQL**: Comprehensive language for relational database management. **CREATE statement**: Main SQL command for data definition. **SQL schema**: Identified by a schema name. Includes an authorization identifier and descriptors for each element. Schema elements include Tables, constraints, views, domains, and other constructs. **Catalog:** Named collection of schemas. **Specifying a new relation** 1) Provide name of table 2) Specify attributes, their types and initial constraints. **Base tables (relations):** Relation and its tuples are actually created and stored as a file by the DBM. SQL Standards: core specification and specialized extensions. **Virtual relations (views)**: Created through the CREATE VIEW statement. Do not correspond to any physical file. **Basic data types**: **Numeric data types**: INTEGER, INT, and SMALLINT, FLOAT or REAL, and DOUBLE PRECISION. **Character-string**: CHAR(n), CHARACTER(n), VARCHAR(n), CHAR VARYING(n), CHARACTER VARYING(n). **Bit-string**: BIT(n), BIT VARYING(n). **Boolean**: TRUE or FALSE or NULL. **DATE**: YYYY-MM-DD (Mapping Functions avalaible in DBMS). **Timestamp**: Includes the DATE and TIME also TIMEZONE. **INTERVAL**: increment or decrement an absolute value of a date, time, or timestamp. **Domain**: Name used with the attribute specification. Makes it easier to change the data type for a domain that is used by numerous attributes. Improves schema readability. Example: CREATE DOMAIN SSN\_TYPE AS CHAR(9). **UDT's: TYPE**: User Defined Types are supported for object-oriented applications. **Basic Constraints**: **Key constraint**: A primary key value cannot be duplicated. Specifies one or more attributes that make up the primary key of a relation. UNIQUE clause Specifies alternate (secondary) keys (called CANDIDATE keys in the relational model). **Entity Integrity Constraint**: A primary key value cannot be null. **Referential integrity constraints**: “foreign key” must have a value that is already present as a primary key, or may be null. **PRIMARY KEY clause**: Specifies one or more attributes that make up the primary key of a relation. **UNIQUE clause**: Specifies alternate (secondary) keys (called CANDIDATE keys in the relational model). **FOREIGN KEY clause**: Default operation is reject update on violation. Attach referential triggered action clause a. Options include SET NULL, CASCADE, and SET DEFAULT b. Action taken by the DBMS for SET NULL or SET DEFAULT is the same for both ON DELETE and ON UPDATE c. CASCADE option suitable for “relationship" relations**.** **CHECK clauses** at the end of a CREATE TABLE statement- Apply to each tuple individually. CHECK (Dept\_create\_date <= Mgr\_start\_date); **Attribute Constraints**: DEFAULT <value> , NULL (NOT NULL), CHECK (Apply to each tuple individually). **Using the keyword CONSTRAINT**: a. Name a constraint b. Useful for later altering. **SELECT statement**: One basic statement for retrieving information from a database. **Projection attributes** are: a) Attributes whose values are to be retrieved b) **Selection condition**: Boolean condition that must be true for any retrieved tuple. Selection conditions include join conditions when multiple relations are involved. **SELECT <attribute list> FROM <Table List> WHERE <condition> ORDER BY <attribute list>** **Select**: list of attribute names to be received by the query **From**: the tables that these attributes with be retrieved from **Where**: conditional Boolean expression to identify certain tuples to be retrieved (optional) **Order by**: attribute list to order the result by (optional). **SQL differ from relation**: sql allows a table (relation) to have 2 or more tuples that are identical in all their attri values. Sql tables is not set of tuples becoz a set does not allow 2 identical members. It is a multiset of tuples. Some SQL relations are constrained to be sets because a key constraint has been declared or becoz of DISTINCT option has been used in SELECT stmt. In relation is a set of tuples that is same values are not allowed for any tuple. **Query 0**. Retrieve the birth date and address of the employee(s) whose name is ‘John B. Smith’. **SELECT BDate, Address FROM EMPLOYEE WHERE FName = 'John' AND Minit = 'B' AND LName = 'Smith'; Query 1.** Retrieve the name and address of all employees who work for the ‘Research’ department. **SELECT Fname, Minit, Lname, Address FROM EMPLOYEE AS E, DEPARTMENT AS D WHERE E.Dno = D.Dnumber AND D.DName = 'Research'; Query 2**. For every project located in ‘Stafford’, list the project number, the controlling department number, and the department manager’s last name, address, and birth date. **SELECT P.Pnumber, P.Dnum, E.Lname, E.Address, E.Bdate FROM EMPLOYEE AS E, PROJECT AS P, DEPARTMENT AS D WHERE P.Plocation = 'Stafford' AND P.Dnum = D.Dnumber AND D.Mgr\_ssn = E.Ssn; Query 8**. For each employee, retrieve the employee’s first and last name and the first and last name of his or her immediate supervisor. **SELECT E.Fname, E.Lname, S.Fname, S.Lname FROM EMPLOYEE AS E, EMPLOYEE AS S WHERE E.Super\_ssn = S.Ssn; Queries 9 and 10**. Select all EMPLOYEE Ssns (Q9) and all combinations of EMPLOYEE Ssn and DEPARTMENT Dname (Q10) in the database. **SELECT Ssn FROM EMPLOYEE; SELECT E.Ssn, D.Dname FROM EMPLOYEE AS E, DEPARTMENT AS D; Query 11**. Retrieve the salary of every employee (Q11) and all distinct salary values (Q11A). **SELECT E.Salary FROM EMPLOYEE AS E; SELECT DISTINCT E.Salary FROM EMPLOYEE AS E; Query 4**. Make a list of all project numbers for projects that involve an employee whose last name is ‘Smith’, either as a worker or as a manager of the department that controls the project. *Method 1*: **(SELECT DISTINCT P.Pnumber FROM PROJECT AS P, EMPLOYEE AS E, DEPARTMENT AS D WHERE E.Lname = 'Smith' AND P.Dnum = D.Dnumber AND D.Mgr\_ssn = E.Ssn) UNION (SELECT DISTINCT P.Pnumber FROM PROJECT AS P, EMPLOYEE AS E, WORKS\_ON AS W WHERE E.Lname = 'Smith' AND P.Pnumber = W.Pno AND W.Essn = E.Ssn);** *Method 2*: **SELECT DISTINCT Pnumber FROM PROJECT WHERE Pnumber IN (SELECT Pnumber FROM PROJECT AS P, DEPARTMENT AS D, EMPLOYEE AS E WHERE P.Dnum = D.Dnumber AND D.Mgr\_ssn = E.Ssn AND E.Lname = 'Smith') OR Pnumber IN (SELECT Pno FROM WORKS\_ON AS W, EMPLOYEE AS E WHERE W.Essn = E.Ssn AND E.Lname = 'Smith'); Query 12**. Retrieve all employees whose address is in Houston, Texas. **SELECT Fname,Lname FROM EMPLOYEE WHERE Adress LIKE ‘%Houston,TX%’; Query 12A**. Find all employees who were born during the 1950s. **SELECT Fname,Lname FROM EMPLOYEE WHERE Bdate LIKE ‘- - 7 - - - - - - -’; Query 13**. Show the resulting salaries if every employee working on the ‘ProductX’ project is given a 10% raise. **SELECT E.Fname, E.Lname,(E.Salary\*1.1) AS Increased\_Salary FROM EMPLOYEE AS E, WORKS\_ON AS W, PROJECT AS P WHERE E.Ssn = W.Essn AND W.Pno = P.Pnumber AND P.Pname = ‘ProductX’; Query 14**. Retrieve all employees in department 5 whose salary is between $30,000 and $40,000. **SELECT \* FROM EMPLOYEE WHERE Dno = 5 AND (Salary BETWEEN 30000 AND 40000); Query 15**. Retrieve a list of employees and the projects they are working on, ordered by department and, within each department, ordered alphabetically by last name, then first name. **SELECT D.Dname,E.Lname,E.Fname,P.Pname FROM DEPARTMENT AS D, EMPLOYEE AS E, WORKS\_ON AS W, PROJECT AS P WHERE D.Dnumber = E.Dno AND E.Ssn = W.Essn AND W.Pno = P.Pnumber ORDER BY D.Dname, E.Lname, E.Fname;**

**Aliases or tuple variables**: Declare alternative relation names. Recommended practice to abbreviate names and to prefix same or similar attribute from multiple tables. asterisk (\*):Retrieve all the attribute values of the selected tuples. \* can be prefixed by the relation name. **DISTINCT**: Only distinct tuples should remain in the result. **Set operations**: UNION, EXCEPT (difference), INTERSECT. **Corresponding multiset operations**: UNION ALL, EXCEPT ALL, INTERSECT ALL. Type compatibility is needed for these operations to be valid. **Query 4**. Make a list of all project numbers for projects that involve an employee whose last name is ‘Smith’, either as a worker or as a manager of the department that controls the project. **Q4A: ( SELECT DISTINCT Pnumber FROM PROJECT, DEPARTMENT, EMPLOYEE WHERE Dnum = Dnumber AND Mgr\_ssn = Ssn AND Lname = ‘Smith’ ) UNION ( SELECT DISTINCT Pnumber FROM PROJECT, WORKS\_ON, EMPLOYEE WHERE Pnumber = Pno AND Essn = Ssn AND Lname = ‘Smith’ )**; **LIKE (comparison operator)**: Used for string pattern matching. % replaces an arbitrary number of zero or more characters. underscore (\_) replaces a single character. **BETWEEN** is also a comparison operator. **WHERE(Salary BETWEEN 30000 AND 40000) AND Dno = 5**; **Query 13.** Show the resulting salaries if every employee working on the ‘ProductX’ project is given a 10 percent raise. **Q13: SELECT E.Fname, E.Lname, 1.1 \* E.Salary AS Increased\_sal FROM EMPLOYEE AS E, WORKS\_ON AS W, PROJECT AS P WHERE E.Ssn = W.Essn AND W.Pno**; **ORDER BY clause**: DESC result in descending order of values. ASC result in ascending order. Typically placed at the end of the query. **INSERT** typically inserts a tuple (row) in a relation (table). INSERT is used for bulk-loading. **UPDATE** may update a number of tuples (rows) in a relation (table) that satisfy the condition. DELETE may also update a number of tuples (rows) in a relation (table) that satisfy the condition. **Additional Features of SQL**: CREATE INDEX, Transaction control commands; granting and revoking of privileges; Constructs for creating trigger; User Defined Types (CREATE TYPE); XML and OLAP.

**Chapter 7: Query 18**. Retrieve the names of all employees who do not have supervisors. **SELECT E.Fname,E.Lname FROM EMPLOYEE AS E WHERE E.Super\_Ssn IS NULL; Query**: Retrieve the names of employees whose salary is greater than the salary of all the employees in department 5: **SELECT Fname, Lname FROM EMPLOYEE WHERE Salary > ALL (SELECT Salary FROM EMPLOYEE WHERE Dno = 5); Query 16**. Retrieve the name of each employee who has a dependent with the same first name and is the same sex as the employee. *Method 1*: **SELECT E.Fname, E.Lname FROM EMPLOYEE AS E WHERE E.Ssn IN (SELECT D.Essn FROM DEPENDENT AS D WHERE E.Fname = D.Dependent\_name AND E.Sex = D.Sex);** *Method 2*: **SELECT E.Fname, E.Lname FROM EMPLOYEE AS E, DEPENDENT AS D WHERE E.Ssn = D.Essn AND E.Sex = D.Sex AND E.Fname = D.Dependent\_name;** *Method 3*: **SELECT E.Fname, E.Lname FROM EMPLOYEE AS E WHERE EXISTS (SELECT D.Essn FROM DEPENDENT AS D WHERE E.Fname = D.Dependent\_name AND E.Sex = D.Sex); Query 6**. Retrieve the names of employees who have no dependents. **SELECT Fname, Lname FROM EMPLOYEE WHERE NOT EXISTS (SELECT \* FROM DEPENDENT WHERE Ssn = Essn); Query 7**. List the names of managers who have at least one dependent. **SELECT Fname, Lname FROM EMPLOYEE WHERE EXISTS ( SELECT \* FROM DEPENDENT WHERE Ssn = Essn ) AND EXISTS ( SELECT \* FROM DEPARTMENT WHERE Ssn = Mgr\_ssn ); Query 17**. Retrieve the Social Security numbers of all employees who work on project numbers 1, 2, or 3. **SELECT DISTINCT Essn FROM WORKS\_ON WHERE Pno IN (1, 2, 3); Query 19**. Find the sum of the salaries of all employees, the maximum salary, the minimum salary, and the average salary. **SELECT SUM (Salary), MAX (Salary), MIN (Salary), AVG (Salary) FROM EMPLOYEE; Query 20**. Find the sum of the salaries of all employees of the ‘Research’ department, as well as the maximum salary, the minimum salary, and the average salary in this department. **SELECT SUM (Salary), MAX (Salary), MIN (Salary), AVG (Salary) FROM (EMPLOYEE JOIN DEPARTMENT ON Dno = Dnumber) WHERE Dname = ‘Research’; Queries 21 and 22**. Retrieve the total number of employees in the company (Q21) and the number of employees in the ‘Research’ department (Q22). **SELECT COUNT (\*) FROM EMPLOYEE; SELECT COUNT (\*) FROM EMPLOYEE, DEPARTMENT WHERE DNO = DNUMBER AND DNAME = ‘Research’; Query 23**. Count the number of distinct salary values in the database. **SELECT COUNT (DISTINCT Salary) FROM EMPLOYEE; Query 5**: retrieve the names of all employees who have two or more Dependents. **SELECT Lname, Fname FROM EMPLOYEE WHERE ( SELECT COUNT (\*) FROM DEPENDENT WHERE Ssn = Essn ) > = 2; Query 24**. For each department, retrieve the department number, the number of employees in the department, and their average salary. **SELECT Dno, COUNT (\*), AVG (Salary) FROM EMPLOYEE GROUP BY Dno; Query 25**. For each project, retrieve the project number, the project name, and the number of employees who work on that project. **SELECT Pnumber, Pname, COUNT (\*) FROM PROJECT, WORKS\_ON WHERE Pnumber = Pno GROUP BY Pnumber, Pname; Query 26**. For each project on which more than two employees work, retrieve the project number, the project name, and the number of employees who work on the project. **SELECT Pnumber, Pname, COUNT (\*) FROM PROJECT, WORKS\_ON WHERE Pnumber = Pno GROUP BY Pnumber, Pname HAVING COUNT (\*) > 2; Query 27**. For each project, retrieve the project number, the project name, and the number of employees from department 5 who work on the project. **SELECT Pnumber, Pname, COUNT (\*) FROM PROJECT, WORKS\_ON, EMPLOYEE WHERE Pnumber = Pno AND Ssn = Essn AND Dno = 5 GROUP BY Pnumber, Pname; Query 28**. For each department that has more than five employees, retrieve the department number and the number of its employees who are making more than $40,000. *Method 1*: **SELECT Dno, COUNT (\*) FROM EMPLOYEE WHERE Salary>40000 AND Dno IN ( SELECT Dno FROM EMPLOYEE GROUP BY Dno HAVING COUNT (\*) > 5)GROUP BY Dno;** *Method 2:* **WITH BIGDEPTS (Dno) AS ( SELECT Dno FROM EMPLOYEE GROUP BY Dno HAVING COUNT (\*) > 5) SELECT Dno, COUNT (\*)FROM EMPLOYEE WHERE Salary>40000 AND Dno IN BIGDEPTS GROUP BY Dno;**

Retrieve the name of each employee who works on all the projects controlled by department number 5. **SELECT Fname, Lname FROM EMPLOYEE WHERE NOT EXISTS ( ( SELECT Pnumber FROM PROJECT WHERE Dnum = 5) EXCEPT ( SELECT Pno FROM WORKS\_ON WHERE Ssn = Essn) );** Join: **SELECT Fname, Lname, Address FROM (EMPLOYEE JOIN DEPARTMENT ON Dno = Dnumber) WHERE Dname = ‘Research’;** Natural Join: **SELECT Fname, Lname, Address FROM (EMPLOYEE NATURAL JOIN (DEPARTMENT AS DEPT (Dname, Dno, Mssn, Msdate))) WHERE Dname = ‘Research’;** Left Outer Join**: SELECT E.Lname AS Employee\_name, S.Lname AS Supervisor\_name FROM (EMPLOYEE AS E LEFT OUTER JOIN EMPLOYEE AS S ON E.Super\_ssn = S.Ssn);** multiway join**: SELECT Pnumber, Dnum, Lname, Address, Bdate FROM ((PROJECT JOIN DEPARTMENT ON Dnum = Dnumber) JOIN EMPLOYEE ON Mgr\_ssn = Ssn) WHERE Plocation = ‘Stafford’;** Case: **UPDATE EMPLOYEE SET Salary = CASE WHEN Dno = 9THEN Salary + 2000 WHEN Dno = 8THEN Salary + 1500 WHEN Dno = 1THEN Salary + 1000 ELSE Salary + 0 END;** Recursive: **WITH RECURSIVE SUP\_EMP (SupSsn, EmpSsn) AS (SELECT SupervisorSsn, Ssn FROM EMPLOYEE UNION SELECT E.Ssn, S.SupSsn FROM EMPLOYEE AS E, SUP\_EMP AS S WHERE E.SupervisorSsn = S.EmpSsn) SELECT \* FROM SUP\_EMP;**

Assertion: **CREATE ASSERTION SALARY\_CONSTRAINT CHECK ( NOT EXISTS ( SELECT \* FROM EMPLOYEE E, EMPLOYEE M, DEPARTMENT D WHERE E.Salary>M.Salary AND E.Dno = D.Dnumber AND D.Mgr\_ssn = M.Ssn ) );**

Trigger: **CREATE TRIGGER SALARY\_VIOLATION BEFORE INSERT OR UPDATE OF Salary, Supervisor\_ssn ON EMPLOYEE FOR EACH ROW WHEN (NEW.SALARY > ( SELECT Salary FROM EMPLOYEE WHERE Ssn = NEW. Supervisor\_Ssn)) INFORM\_SUPERVISOR (NEW.Supervisor.Ssn, New.Ssn)**

Meanings of NULL a. Unknown value b. Unavailable or withheld value c. Not applicable attribute. SQL uses a three-valued logic: TRUE, FALSE & UNKNOWN (like Maybe). **AND:** TRUE UNKNOWN = UNKNOWN, FALSE UNKNOWN = FALSE, UNKNOWN TRUE = UNKNOWN, UNKNOWN FALSE = FALSE, UNKNOWN UNKNOWN = UNKNOWN **OR:** TRUE UNKNOWN = TRUE, FALSE UNKNOWN = UNKNOWN, UNKNOWN TRUE = TRUE, UNKNOWN FALSE = UNKNOWN, UNKNOWN UNKNOWN = UNKNOWN.

**Nested Queries:** Complete select-from-where blocks within WHERE clause of another query. Outer query and nested subqueries. **Comparison operator IN** - Compares value v with a set (or multiset) of values V. Evaluates to TRUE if v is one of the elements in V to compare a single value v = **ANY (or = SOME)** operator. Returns TRUE if the value v is equal to some value in the set V and is hence equivalent to IN. ALL: value must exceed all values from nested query. Avoid potential errors and ambiguities - Create tuple variables (aliases) for all tables referenced in SQL query. **Correlated nested query** - Evaluated once for each tuple in the outer query. **EXISTS function**: Check whether the result of a correlated nested query is empty or not. They are Boolean functions that return a TRUE or FALSE result. EXISTS and NOT EXISTS: Typically used in conjunction with a correlated nested query. SQL function **UNIQUE(Q)**: Returns TRUE if there are no duplicate tuples in the result of query Q. To achieve the “for all” (universal quantifier - we use double negation /NOT EXISTS and **EXCEPT. AS :** Rename any attribute that appears in the result of a query. Joined table: Permits users to specify a table resulting from a join operation in the FROM clause of a query. JOIN may also be called **INNER JOIN**. Specify different types of join: **NATURAL JOIN**. Various types of **OUTER JOIN (LEFT, RIGHT, FULL). NATURAL JOIN**: No join condition specified. Is equivalent to an implicit **EQUIJOIN** condition for each pair of attributes with same name from R and S. Rename attributes of one relation so it can be joined with another. **INNER JOIN**: Default type of join. Tuple is included in the result only if a matching tuple exists in the other relation. **LEFT OUTER JOIN:** Every tuple in left table must appear in result. If no matching tuple, Padded with NULL values for attributes of right table. **RIGHT OUTER JOIN:** Every tuple in right table must appear in result. If no matching tuple, Padded with NULL values for attributes of left table. **FULL OUTER JOIN** – combines result if LEFT and RIGHT OUTER JOIN. **Aggregate Functions in SQL**: Used to summarize information from multiple tuples into a single-tuple summary. **COUNT, SUM, MAX, MIN, and AVG**. Aggregate functions can be used in the SELECT clause or in a **HAVING clause**. NULL values are discarded. **Grouping**: Create subgroups of tuples before summarizing. To select entire groups, HAVING clause is used. SOME returns true if at least one element in the collection is TRUE (similar to OR). ALL returns true if all of the elements in the collection are TRUE (similar to AND). **Group BY Clause**: Partition relation into subsets of tuples. Based on grouping attribute(s). Apply function to each such group independently. Specifies grouping attributes. If the grouping attribute has NULL as a possible value, then a separate group is created for the null value. HAVING clause: Provides a condition to select or reject an entire group. WHERE clause applies tuple by tuple whereas HAVING applies to entire group of tuples. **WITH**: allows a user to define a table that will only be used in a particular query. Used for convenience to create a temporary “View” and use that immediately in a query. way of looking a step-by-step query. **CASE**: Used when a value can be different based on certain conditions. Can be used in any part of an SQL query where a value is expected. Applicable when querying, inserting or updating tuples. Recursive Queries: recursive relationship between tuples of the same type. **CREATE ASSERTION**: Specify additional types of constraints outside scope of built-in relational model constraints. Specify a query that selects any tuples that violate the desired condition. Use only in cases where it goes beyond a simple **CHECK** which applies to individual attributes and domains. **CREATE TRIGGER**: Specify automatic actions that database system will perform when certain events and conditions occur. Used to monitor the database. three components which make it a rule for an “active database “. Event(s), Condition & Action. Concept of a **view in SQL**: Single table derived from other tables called the defining tables. Considered to be a virtual table that is not necessarily populated. Give table name, list of attribute names, and a query to specify the contents of the view. View is always up-to-date. Responsibility of the DBMS and not the user. DROP VIEW command - Dispose of a view.

**Strategy 1:** Query modification approach: Compute the view as and when needed. Do not store permanently. Modify view query into a query on underlying base tables. Disadvantage: inefficient for views defined via complex queries that are time-consuming to execute.

**Strategy 2:** View materialization. Physically create a temporary view table when the view is first queried. Keep that table on the assumption that other queries on the view will follow. Requires efficient strategy for automatically updating the view table when the base tables are updated.

**Strategy 3:** Incremental update strategy for materialized views. DBMS determines what new tuples must be inserted, deleted, or modified in a materialized view table.

**Multiple ways to handle materialization**: immediate update strategy - updates a view as soon as the base tables are changed. lazy update strategy - updates the view when needed by a view query. periodic update strategy - updates the view periodically (in the latter strategy, a view query may get a result that is not up-to-date). This is commonly used in Banks, Retail store operations, etc.

**View Update** - Update on a view defined on a single table without any aggregate functions. Can be mapped to an update on underlying base table- possible if the primary key is preserved in the view.

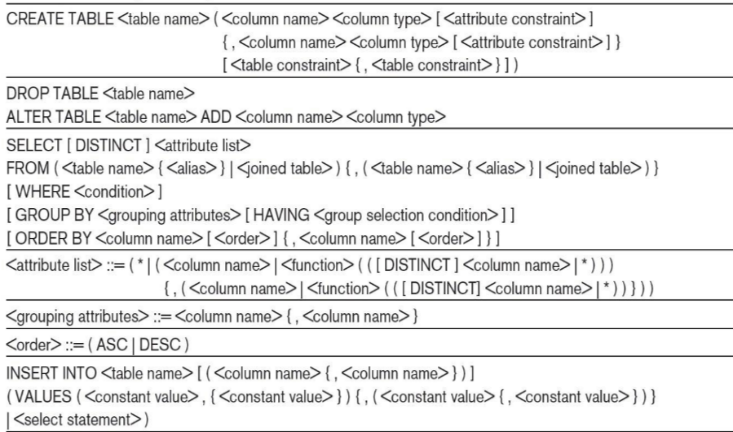
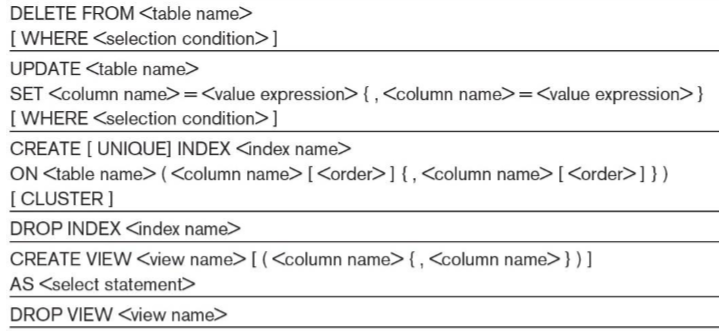
**Clause WITH CHECK OPTION** - Must be added at the end of the view definition if a view is to be updated to make sure that tuples being updated stay in the view. In-line view: Defined in the FROM clause of an SQL query (e.g., we saw its used in the WITH example). Views can be used to hide certain attributes or tuples from unauthorized users.

**Schema evolution commands** - DBA may want to change the schema while the database is operational. Does not require recompilation of the database schema.

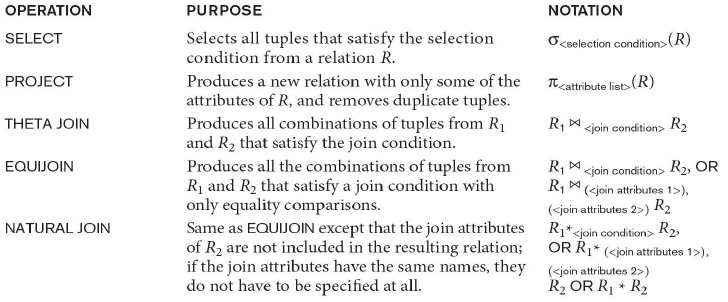
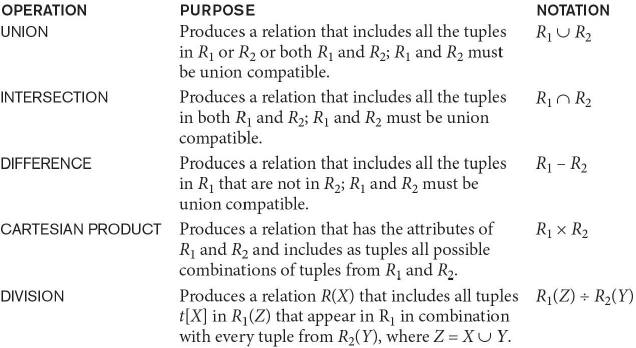
**DROP command**: Used to drop named schema elements, such as tables, domains, or constraint. Drop behavior options: CASCADE and RESTRICT. DROP SCHEMA COMPANY CASCADE; This removes the schema and all its elements including tables, views, constraints, etc.

Alter table actions include: Adding or dropping a column (attribute). Changing a column definition. Adding or dropping table constraints. ALTER TABLE COMPANY.EMPLOYEE ADD COLUMN Job VARCHAR(12);

drop a column: Choose either CASCADE or RESTRICT. Default values can be dropped and altered.

Relational algebra is the basic set of operations for the relational model. The result of an operation is a new relation, which may have been formed from one or more input relations. Relational Algebra consists of several groups of operations ◼ Unary Relational Operations SELECT(sigma), PROJECT(pi), RENAME(rho) ◼Relational Algebra Operations From Set Theory  UNION,INTERSECTION,DIFFERENCE(orMINUS, –),CARTESIAN PRODUCT(x) ◼ Binary Relational OperationsJOIN,DIVISION ◼Additional Relational OperationsOUTER JOINS,OUTER UNION,AGGREGATE FUNCTIONS(SUM, COUNT, AVG, MIN, MAX)

An internal data structure to represent a query. Nodes stand for operations like selection, projection, join, renaming, division, …. Leaf nodes represent base relations. Algebraic Query Optimization consists of rewriting the query or modifying the query tree into an equivalent tree.

DNO ℱCOUNT SSN, AVERAGE Salary (EMPLOYEE)

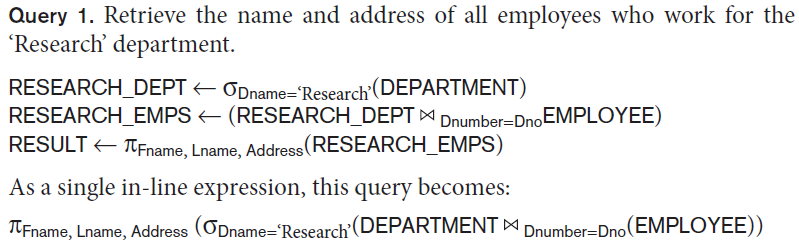
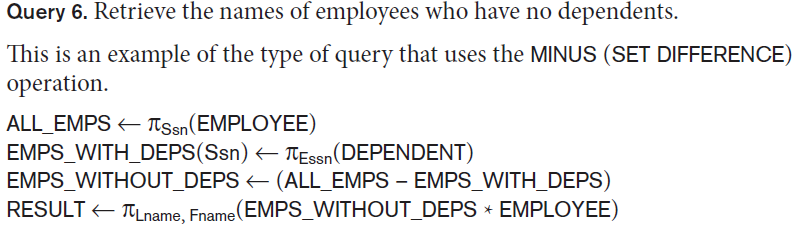
Another type of operation that, in general, cannot be specified in the basic original relational algebra is recursive closure. This operation is applied to a recursive relationship.

LEFT JOIN R =|X| S

RIGHT JOIN R |X|= S

FULL JOIN R =|X|= S

OUTER UNION - This operation will take the union of tuples in two relations R(X, Y) and S(X, Z) that are partially compatible, meaning that only some of their attributes, say X, are type compatible.



SELECT is commutative



PROJECT creates a vertical partitioning. PROJECT is not commutative. The project operation removes any duplicate tuples ◼ This is because the result of the project operation must be a set of tuples. The number of tuples in the result of projection pi<list>(R) is always less or equal to the number of tuples in R. pi <list1> (pi <list2> (R) ) = pi <list1> (R) as long as <list2> contains the attributes in <list1>.

rhoS (B1, B2, …, Bn )(R) changes both: ◼ the relation name to S, and ◼ the column (attribute) names to B1, B1, …..Bn

RESULT (F, M, L, S, B, A, SX, SAL, SU, DNO)<- rho RESULT (F.M.L.S.B,A,SX,SAL,SU, DNO)(DEP5\_EMPS)

UNION Operation: Duplicate tuples are eliminated ◼ The two operand relations R and S must be “type compatible” (or UNION compatible) ◼ R and S must have same number of attributes ◼ Each pair of corresponding attributes must be type compatible (have same or compatible domains)

Relational Algebra Operations from Set Theory : R1(A1, A2, ..., An) and R2(B1, B2, ..., Bn) are type compatible if: ◼ they have the same number of attributes, and ◼ the domains of corresponding attributes are type compatible (i.e. dom(Ai)=dom(Bi) for i=1, 2, ..., n).

Union, intersection, SET DIFFERENCE – type compatible

Minus not commutative

CARTESIAN (or CROSS) PRODUCT Operation ◼ This operation is used to combine tuples from two relations in a combinatorial fashion. ◼ Denoted by R(A1, A2, . . ., An) x S(B1, B2, . . ., Bm) ◼ Result is a relation Q with degree n + m attributes: ◼ Q(A1, A2, . . ., An, B1, B2, . . ., Bm), in that order. ◼ The resulting relation state has one tuple for each combination of tuples—one from R and one from S. ◼ Hence, if R has nR tuples (denoted as |R| = nR ), and S has nS tuples, then R x S will have nR \* nS tuples. ◼ The two operands do NOT have to be "type compatible”

JOIN Operation (denoted by ) ◼ The sequence of CARTESIAN PRODECT followed by SELECT is used quite commonly to identify and select related tuples from two relations ◼ A special operation, called JOIN combines this sequence into a single operation ◼ This operation is very important for any relational database with more than a single relation, because it allows us combine related tuples from various relations ◼ The general form of a join operation on two relations R(A1, A2, . . ., An) and S(B1, B2, . . ., Bm) is: R <join condition>S ◼ where R and S can be any relations that result from general relational algebra expressions.

Relational algebra is the basic set of operations for the relational model . The result of an operation is a new relation, which may have been formed from one or more input relations . Relational Algebra consists of several groups of operations ? Unary Relational Operations ?SELECT(sigma) ?PROJECT(pi) ?RENAME(rho) ?Relational Algebra Operations From Set Theory ? UNION,INTERSECTION,DIFFERENCE(orMINUS, –)?CARTESIAN PRODUCT(x) ? Binary Relational Operations ?JOIN ?DIVISION ?Additional Relational Operations ?OUTER JOINS,OUTER UNION ? AGGREGATE FUNCTIONS(SUM, COUNT, AVG, MIN, MAX) SELECT - sigma <condition> (R) - is commutative PROJECT- pi <attributes\_list> (R) PROJECT is not commutative ? pi <list1> (pi <list2> (R) ) = pi <list1> (R) as long as <list2> contains the attributes in <list1> rho S (B1, B2, …, Bn )(R) changes both: ? the relation name to S, and ? the column (attribute) names to B1, B1, …..Bn ? rho S(R) changes: ? the relation name only to S ? rho(B1, B2, …, Bn )(R) changes: ? the column (attribute) names only to B1, B1, …..Bn RESULT (F, M, L, S, B, A, SX, SAL, SU, DNO)? ? RESULT (F.M.L.S.B,A,SX,SAL,SU, DNO)(DEP5\_EMPS). The two operand relations R and S must be “type compatible” (or UNION compatible) ? R and S must have same number of attributes ? Each pair of corresponding attributes must be type compatible (have same or compatible domains). Notice that both union and intersection are commutative operations; ? Both union and intersection can be treated as n-ary operations applicable to any number of relations as both are associative operations; ? The minus operation is not commutative; that is, in general ? R – S ? S – R

JOIN - R |X| <join condition>S The general case of JOIN operation is called a Theta-join: R |X|theta S

EQUIJOIN - common use of join involves join conditions with equality comparisons only. In the result of an EQUIJOIN we always have one or more pairs of attributes (whose names need not be identical) that have identical values in every tuple.

NATURAL JOIN - The standard definition of natural join requires that the two join attributes, or each pair of corresponding join attributes, have the same name in both relations. The set of operations including SELECT,PROJECT,UNION,DIFFERENCE,RENAME,CARTESIAN PRODUCT is called a complete set because any other relational algebra expression can be expressed by a combination of these five operations.

◼ R int S = (R u S ) – ((R - S) u (S - R)) ◼ R |X| <join condition>S = sigma <join condition> (R X S)

DIVISION - For a tuple t to appear in the result T of the DIVISION, the values in t must appear in R in combination with every tuple in S.