**Structured Query Language DML** (Last updated 3/30/2020)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Structured Query Language (SQL)**  E.F. Codd's work on Alpha evolved into SQL.  SQL has three major components:  **Data Definition Language** – create/drop tables, indexes, and views  **Data Control Language** – grant/revoke privileges to administer databases and perform operations on tables (Select, Insert, Delete, Update)  **Data Manipulation Language** – perform operations (Select, Insert, Delete, Update) on tables and views of tables.  Each of the DML operations can perform the operation for one or more rows.  **Comments** in SQL begin with **two dashes.** | Consider the following relations:   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | Student   |  |  | | --- | --- | | **studentNr** | **classif** | | 100 | FR | | 200 | SO | | 300 | JR | | 400 | SR | | 500 | FR |   Course   |  |  | | --- | --- | | **courseNr** | **title** | | CS 1713 | Intro to CS | | MAT 1224 | Calculus I | | ENG 1043 | English I | | HIS 1033 | History of the World Part I | | Enrollment   |  |  |  |  | | --- | --- | --- | --- | | **studentNr** | **courseNr** | **midTerm** | **final** | | 100 | CS 1713 | 95 | 90 | | 100 | MAT 1224 | 100 | 100 | | 100 | ENG 1043 | 85 | 85 | | 100 | HIS 1033 | 80 | 85 | | 200 | CS 1713 | 91 | 90 | | 200 | MAT 1224 | 50 | 80 | | 300 | CS 1713 | 65 | 80 | | 500 | CS 1713 | 100 | 80 | | 500 | HIS 1033 | 95 | 90 | | |
| **Example DDL**  We will discuss the DDL in detail in the next document. | **Example 1: create table**  -- create the Enrollment table with just the information above  CREATE TABLE `edudb`.`Enrollment` (  `studentNr` INT NOT NULL,  `courseNr` CHAR(7) NOT NULL,  `midTerm` INT(3) DEFAULT 0,  `final` INT(3) DEFAULT 0,  PRIMARY KEY (`studentNr`, `courseNr`)); |
| **SQL Insert Statement**  There are several forms of an INSERT statement:   * Single row * Multiple row * Inserting Rows using a Query |  |
| **Single Row Insert**  We can either explicitly list the columns or default it to the order of the columns in the system catalog (which is based on the create table statement's column order).  INSERT INTO *database.tableName* (*columnList*)  VALUES(*valueList*);  Notes:  1. Any columns not listed will be defaulted to either NULL or the default values specified in the DDL. For more information, see the SQL DDL notes.  2. The database, table names, and column names **are** case sensitive.  3. SQL statements are terminated with semicolons. | **Example 2: inserting a single row into Enrollment.**  -- I2.1 specify all the columns  INSERT INTO Enrollment (studentNr, courseNr, midTerm, final)  VALUES(100, "CS 1713", 95, 80);  -- I2.2 Assume the DDL defaulted midTerm and final to zero, we can leave  -- them out of the list and the specified defaults will happen.  INSERT INTO Enrollment (studentNr, courseNr)  VALUES(600, "CS 3743");  -- I2.3 default the columns to those in the catalog  INSERT INTO Enrollment VALUES(700, "CS 1713", 75, 80);  mysql> select \* from Enrollment;  +-----------+----------+---------+-------+  | studentNr | courseNr | midTerm | final |  +-----------+----------+---------+-------+  | 100 | CS 1713 | 95 | 90 |  | 600 | CS 3743 | 0 | 0 |  | 700 | CS 1713 | 75 | 80 |  +-----------+----------+---------+-------+ |
| **Multiple Row Insert**  This continues on the syntax for inserting a single row:  INSERT INTO *database.tableName* (*columnList*)  VALUES(*valueList*), (*valueList*), …;  We can simply list multiple tuples. | **Example 3: inserting multiple rows into Enrollment.**  INSERT INTO Enrollment (studentNr, courseNr, midTerm, final)  VALUES(201, "CS 1713", 100, 90),  (201, "MAT1224", 0, 0),  (201, "ENG1013", 50, 50);  mysql> select \* from Enrollment;  +-----------+----------+---------+-------+  | studentNr | courseNr | midTerm | final |  +-----------+----------+---------+-------+  | 100 | CS 1713 | 95 | 90 |  | 201 | CS 1713 | 100 | 90 |  | 201 | ENG1013 | 50 | 50 |  | 201 | MAT1224 | 0 | 0 |  | 600 | CS 3743 | 0 | 0 |  | 700 | CS 1713 | 75 | 80 |  +-----------+----------+---------+-------+ |
| **Inserting Rows Using a Query**  The results from a query can be used to populate a table.  INSERT INTO *database.tableName* (*columnList*)  SELECT *columnList*  FROM *tableList*  WHERE *condition*; | **Example 4: inserting rows using a query's result**  **--** Copy rows from student 201 to create student 444**.**  INSERT INTO Enrollment (studentNr, courseNr, midTerm, final)  SELECT 444, e.courseNr, e.midTerm, e.final  FROM Enrollment e  WHERE e.studentNr = 201;  mysql> select \* from Enrollment;  +-----------+----------+---------+-------+  | studentNr | courseNr | midTerm | final |  +-----------+----------+---------+-------+  | 100 | CS 1713 | 95 | 90 |  | 201 | CS 1713 | 100 | 90 |  | 201 | ENG1013 | 50 | 50 |  | 201 | MAT1224 | 0 | 0 |  | 444 | CS 1713 | 100 | 90 |  | 444 | ENG1013 | 50 | 50 |  | 444 | MAT1224 | 0 | 0 |  | 600 | CS 3743 | 0 | 0 |  | 700 | CS 1713 | 75 | 80 |  +-----------+----------+---------+-------+ |
| **SQL Delete Statement**  This is a very powerful statement since it allows you to delete many rows from a table. Two forms:  DELETE FROM  *database.tableName*;  This unconditionally deletes all the rows from the specified table.  DELETE FROM  *database.tableName*  WHERE *condition*;  This conditionally deletes the rows from the specified table which satisfy the condition.  MySQL Workbench provides a setting which is defaulted to not allow you to delete rows from a table without specifying the full key. | **Example 5: deleting**  **-- D5.1** Delete all the rows in the Enrollment table  DELETE FROM Enrollment;  **-- D5.2** Delete all the rows in the Enrollment table for student 555.  DELETE FROM Enrollment WHERE studentNr = 555;  **-- D5.3** Delete a particular row from Enrollment: studentNr is 500 and courseNr is "CS 17113".  DELETE FROM Enrollment  WHERE studentNr = 500 AND courseNr="CS 1713"; |
|  | **Example 5: Deleting continued**  **-- D5.4** Delete enrollments for students who are "FR" and flunked the midTerm.  mysql> select \* from Student;  +-----------+---------+  | studentNr | classif |  +-----------+---------+  | 100 | FR |  | 200 | SO |  | 201 | SO |  | 300 | JR |  | 400 | SR |  | 444 | FR |  | 500 | FR |  | 600 | FR |  | 700 | JR |  +-----------+---------+  mysql> select \* from Enrollment;  +-----------+----------+---------+-------+  | studentNr | courseNr | midTerm | final |  +-----------+----------+---------+-------+  | 100 | CS 1713 | 95 | 90 |  | 100 | ENG1043 | 85 | 95 |  | 100 | HIS1033 | 80 | 85 |  | 100 | MAT1224 | 100 | 100 |  | 200 | CS 1713 | 91 | 90 |  | 200 | MAT1224 | 50 | 80 |  | 201 | CS 1713 | 100 | 90 |  | 201 | ENG1013 | 50 | 50 |  | 201 | MAT1224 | 0 | 0 |  | 300 | CS 1713 | 65 | 80 |  | 444 | CS 1713 | 100 | 90 |  | 444 | ENG1013 | 50 | 50 |  | 444 | MAT1224 | 0 | 0 |  | 500 | CS 1713 | 100 | 80 |  | 500 | HIS1033 | 95 | 90 |  | 600 | CS 3743 | 0 | 0 |  | 700 | CS 1713 | 75 | 80 |  +-----------+----------+---------+-------+  DELETE FROM Enrollment  WHERE midTerm < 60  AND EXISTS (  SELECT s.studentNr  FROM Student s  WHERE s.classif = "FR"  AND s.studentNr = Enrollment.studentNr);  **mysql> select \* from Enrollment;**  **+-----------+----------+---------+-------+**  **| studentNr | courseNr | midTerm | final |**  **+-----------+----------+---------+-------+**  **| 100 | CS 1713 | 95 | 90 |**  **| 100 | ENG1043 | 85 | 95 |**  **| 100 | HIS1033 | 80 | 85 |**  **| 100 | MAT1224 | 100 | 100 |**  **| 200 | CS 1713 | 91 | 90 |**  **| 200 | MAT1224 | 50 | 80 |**  **| 201 | CS 1713 | 100 | 90 |**  **| 201 | ENG1013 | 50 | 50 |**  **| 201 | MAT1224 | 0 | 0 |**  **| 300 | CS 1713 | 65 | 80 |**  **| 444 | CS 1713 | 100 | 90 |**  **| 500 | CS 1713 | 100 | 80 |**  **| 500 | HIS1033 | 95 | 90 |**  **| 700 | CS 1713 | 75 | 80 |**  **+-----------+----------+---------+-------+** |
| **SQL Update Statement**  This is a very powerful statement since it allows you to update many rows from a table. Two forms:  UPDATE *database.tableName*  SET *columnNm1=value1*  , *columnNm2=value2*  …;  This unconditionally updates every row in the specified table.  UPDATE *database.tableName*  SET *columnNm1=value1*  , *columnNm2=value2*  …  WHERE *condition*;  This conditionally updates the rows from the specified table which satisfy the condition.  **Note that you only have to specify the columns which need to change.** | **Example 6: updating**  **-- U6.1** Update each of the final exam grades for HIS1033 to be 10% higher.  UPDATE Enrollment  SET final = final \* 1.1  WHERE courseNr = "HIS1033";  **-- U6.2** Update the midterm grade to be the final grade if the final is higher for "CS 1713"  UPDATE Enrollment  SET midTerm = final  WHERE courseNr = "CS 1713"  AND final > midTerm;  mysql> select \* from Enrollment;  +-----------+----------+---------+-------+  | studentNr | courseNr | midTerm | final |  +-----------+----------+---------+-------+  | 100 | CS 1713 | 95 | 90 |  | 100 | ENG1043 | 85 | 95 |  | 100 | HIS1033 | 80 | 85 |  | 100 | MAT1224 | 100 | 100 |  | 200 | CS 1713 | 91 | 90 |  | 200 | MAT1224 | 50 | 80 |  | 201 | CS 1713 | 100 | 90 |  | 201 | ENG1013 | 50 | 50 |  | 201 | MAT1224 | 0 | 0 |  | 300 | CS 1713 | 80 | 80 |  | 444 | CS 1713 | 100 | 90 |  | 500 | CS 1713 | 100 | 80 |  | 500 | HIS1033 | 95 | 90 |  | 700 | CS 1713 | 80 | 80 |  +-----------+----------+---------+-------+ |
|  | **Example 6: updating continued**  **-- U6.3** Update the midterm grade and final grade to 0 if the student is a freshman and the  course is "MAT1224";  UPDATE Enrollment  SET midTerm = 0,  final = 0  WHERE Enrollment.courseNr = "MAT1224"  AND EXISTS (  SELECT s.studentNr  FROM Student s  WHERE s.classif = "FR"  AND s.studentNr = Enrollment.studentNr);  **mysql> select \* from Enrollment;**  **+-----------+----------+---------+-------+**  **| studentNr | courseNr | midTerm | final |**  **+-----------+----------+---------+-------+**  **| 100 | CS 1713 | 95 | 90 |**  **| 100 | ENG1043 | 85 | 95 |**  **| 100 | HIS1033 | 80 | 85 |**  **| 100 | MAT1224 | 0 | 0 |**  **| 200 | CS 1713 | 91 | 90 |**  **| 200 | MAT1224 | 50 | 80 |**  **| 201 | CS 1713 | 100 | 90 |**  **| 201 | ENG1013 | 50 | 50 |**  **| 201 | MAT1224 | 0 | 0 |**  **| 300 | CS 1713 | 80 | 80 |**  **| 444 | CS 1713 | 100 | 90 |**  **| 500 | CS 1713 | 100 | 80 |**  **| 500 | HIS1033 | 95 | 90 |**  **| 700 | CS 1713 | 80 | 80 |**  **+-----------+----------+---------+-------+** |
| **SQL Select Statement**  Forms:  1. SELECT *columnList* FROM *tableList*;  Simple projection of columns.  2. SELECT \* FROM *tableList*;  Projection of all the columns. Most DBAs prefer that you not do this inside programs.  3. SELECT *columnList* FROM *tableList*   WHERE *condition*;  Projection of the specified columns based on rows which match the specified *condition.*  4. SELECT *columnList* FROM *tableList*  WHERE *condition*  ORDER BY *columnList*;  Any of the previous forms can be followed by an ORDER BY to have the results sorted.  5. SELECT *columnList* FROM *tableList*  WHERE *condition*  GROUP BY *columnList*  HAVING condtion;  Discussed below.  Note that duplicate rows in your answer can be removed by specifying SELECT DISTINCT. | **Example 7:** **Projection** - Get the C# for courses in which students are enrolled.  select e.courseNr from Enrollment e;   |  | | --- | | **courseNr** | | CS 1713 | | MAT 1224 | | ENG 1043 | | HIS 1033 |   **Example 8:** **Projection** - Get the ST# and Midterm grades of each student.  select e.studentNr, e.midTerm from Enrollment e;   |  |  | | --- | --- | | **studentNr** | **midTerm** | | 100 | 95 | | 100 | 100 | | 100 | 85 | | 100 | 80 | | 200 | 91 | | 200 | 50 | | 300 | 65 | | 500 | 100 | | 500 | 95 | |
| **Using DISTINCT to Remove Duplicates in a Query**  Unlike Relational Algebra and Relational Calculus, SQL doesn't automatically remove duplicates in a query's result. Instead, you have to specify DISTINCT. | **Example 9:** **Projection** - Get the ST# of each student.  select e.studentNr from Enrollment e;   |  | | --- | | **studentNr** | | 100 | | 100 | | 100 | | 100 | | 200 | | 200 | | 300 | | 500 | | 500 |   To remove the duplicates, we use **DISTINCT:**  select DISTINCT e.studentNr from Enrollment e;   |  | | --- | | **studentNr** | | 100 | | 200 | | 200 | | 300 | | 500 | |
| **Selection**  We specify a condition after the WHERE. | **Example 10:** **Selection -** get enrollment information for students who have a  midterm grade > 90.  select e.studentNr, e.courseNr, e.midTerm, e.final  from Enrollment e  where e.midTerm > 90;   |  |  |  |  | | --- | --- | --- | --- | | **studentNr** | **courseNr** | **midTerm** | **final** | | 100 | CS 1713 | 95 | 90 | | 100 | MAT 1224 | 100 | 100 | | 200 | CS 1713 | 91 | 90 | | 500 | CS 1713 | 100 | 80 | | 500 | HIS 1033 | 95 | 90 |   **Example 11:** **Selection -** get enrollment information of students who made an A on both exams.  select e.studentNr, e.courseNr, e.midTerm, e.final  from Enrollment e  where e.midTerm >= 90 and e.final>=90; |
| **Join** | **Example 12:** **Join -** join the Student and Enrollment relations which have the same ST#  select s.studentNr, s.classif, e.courseNr, e.midTerm, e.final  from Student s, Enrollment e  where e.studentNr = s.StudentNr;   |  |  |  |  |  | | --- | --- | --- | --- | --- | | **studentNr** | **classif** | **courseNr** | **midTerm** | **final** | | 100 | FR | CS 1713 | 95 | 90 | | 100 | FR | MAT 1224 | 100 | 100 | | 100 | FR | ENG 1043 | 85 | 85 | | 100 | FR | HIS 1033 | 80 | 85 | | 200 | SO | CS 1713 | 91 | 90 | | 200 | SO | MAT 1224 | 50 | 80 | | 300 | JR | CS 1713 | 65 | 80 | | 500 | FR | CS 1713 | 100 | 80 | | 500 | FR | HIS 1033 | 95 | 90 | |
| **Not any**  SQL doesn't support the universal quantifier, instead it provides NOT EXISTS.  Recall from Discrete Math:  1. ~a ∨ b = a → b  2. g(x) = ~~g(x)  3. ~∀x (g(x)) = ∃x (~g(x)). | **Example 13: Get the student number of students who are not enrolled in "HIS1033"**  Using relational calculus:  RANGE Student s  RANGE Enrollment e  GET w (s.studentNr):   ∀e ( e.studentNr = s.studentNr → e.courseNr ≠ "HIS1033")  Using ~a ∨ b instead of a → b:  GET w (s.studentNr):   ∀e ( e.studentNr ≠ s.studentNr v e.courseNr ≠ "HIS1033")  Using ~~g(x):  GET w (s.studentNr):   ~~∀e ( e.studentNr ≠ s.studentNr v e.courseNr ≠ "HIS1033")  Using ~∀x (g(x)) = ∃x (~g(x)):  GET w (s.studentNr):   ~∃e (~( e.studentNr ≠ s.studentNr v e.courseNr ≠ "HIS1033"))  Using deMorgans:  GET w (s.studentNr):   ~∃e ( e.studentNr = s.studentNr ^ e.courseNr = "HIS1033")  SQL:  select s.studentNr  from Student s  where NOT EXISTS (  select True  from Enrollment e  where e.studentNr = s.studentNr  and e.courseNr = "HIS1033"); |
| **All** | **Example 14: Get the student number of students who are enrolled in all the courses.**  Using relational calculus:  RANGE Student s  RANGE Course c  RANGE Enrollment e  GET w (s.studentNr): ∀c∃e ( e.studentNr = s.studentNr  ^ e.courseNr = c.courseNr)  Converting to SQL:  GET w (s.studentNr): ~~∀c∃e ( e.studentNr = s.studentNr  ^ e.courseNr = c.courseNr)  GET w (s.studentNr): ~∃c ~(∃e ( e.studentNr = s.studentNr  ^ e.courseNr = c.courseNr))  No need to bring that second ~ inside. Double negative  select s.studentNr  from Student s  where NOT EXISTS (  select True  from Course c  where NOT EXISTS (  select True  from Enrollment e  where e.studentNr = s.studentNr  and e.courseNr = c.courseNr )); |
| **Aggregate Functions - AVG**  **AVG**(*columnName*) - returns the average value in the specified column.  Note that NULL values are NOT considered in the average. If there are no matching rows, AVG returns NULL. | **Example 15: Get the average final grade for CS 1713.**  select AVG(e.final) from Enrollment e  where e.courseNr="CS 1713";  +--------------+  | AVG(e.final) |  +--------------+  | 85.7143 |  +--------------+ |
| **Aggregate Function - COUNT**  **COUNT**(*columnName*) - returns the count of non-NULL values of the specified column based on any specified criteria  **COUNT(\*) -** returns a count of the rows satisfying the criteria regarless of whether it is NULL. | **Example 16: For each student having at least 2 As (>=90) on their finals, show the student's student number and the count.**  select e.studentNr, COUNT(e.final)  from Enrollment e  where e.final >= 90 -- WHERE is for columns not in GROUP BY  group by e.studentNr -- GROUP BY columns must appear in result  having COUNT(e.final)> 1; -- HAVING columns must be aggregate functions  -- or columns in GROUP BY  +-----------+----------------+  | studentNr | COUNT(e.final) |  +-----------+----------------+  | 100 | 2 |  +-----------+----------------+ |
| **Supplier Parts DB Example**  We will use this classical schema in MANY examples.  SUPPLIER(S#, SNAME, STATUS, CITY)  S# - supplier number  SNAME - supplier name(e.g., SMITH, JONES),  STATUS - a number representing the quality of this supplier with the higher number representing a better quality of service (e.g., 10, 20)  PART(P#, PNAME, COLOR, WEIGHT)  P# - part number  PNAME - part name (e.g., NUT)  COLOR - color of the part (e.g., RED)  WEIGHT - weight of the part in grams  SHIPMENT(S#, P#, QUANTITY)  S# - supplier number  P# - part number  QUANTITY - the number of this part shipped by this supplier  **Constraints**: keys are underlined  Note that in most implementations of SQL, the column names S# and C# are invalid. We use them simply to save typing in the examples. Also, we will usually have camel case column names that begin with lower case. | **Supplier Parts DB sample Data**   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **SUPPLIER**   |  |  |  |  | | --- | --- | --- | --- | | **S#** | **SNAME** | **STATUS** | **CITY** | | S1 | SMITH | 20 | LONDON | | S2 | JONES | 10 | PARIS | | S3 | BLAKE | 30 | PARIS | | S4 | FORD | 20 | LONDON | | S5 | ADAMS | 30 | ATHENS |   **PART**   |  |  |  |  | | --- | --- | --- | --- | | **P#** | **PNAME** | **COLOR** | **WEIGHT** | | P1 | NUT | RED | 12 | | P2 | BOLT | GREEN | 17 | | P3 | SCREW | BLUE | 17 | | P4 | SCREW | RED | 14 | | P5 | CAM | BLUE | 12 | | P6 | COG | RED | 19 | | **SHIPMENT**   |  |  |  | | --- | --- | --- | | **S#** | **P#** | **QUANTITY** | | S1 | P1 | 300 | | S1 | P2 | 200 | | S1 | P3 | 400 | | S1 | P4 | 200 | | S1 | P5 | 100 | | S1 | P6 | 100 | | S2 | P1 | 300 | | S2 | P2 | 400 | | S3 | P2 | 200 | | S4 | P2 | 200 | | S4 | P4 | 300 | | S4 | P5 | 400 | | |
| **Simple Retrievals (no qualifying expression)** | **Example SP-1:** Get the part numbers for each of the parts.  select p.P# from PART p;  This returns a subset of the columns -- just the part number.  Result:   |  | | --- | | **P#** | | P1 | | P2 | | P3 | | P4 | | P5 | | P6 |   **Example SP-2:**  Get the supplier number, name, and city for each of the suppliers  select s.S#, s.SNAME, s.CITY from SUPPLIER s;  Result:   |  |  |  | | --- | --- | --- | | S# | SNAME | CITY | | S1 | SMITH | LONDON | | S2 | JONES | PARIS | | S3 | BLAKE | PARIS | | S4 | FORD | LONDON | | S5 | ADAMS | ATHENS | |
| **Simple Qualified Retrievals** | **Example SP-3:**  Get the supplier number of suppliers in Paris with a status > 20.  select s.S# from SUPPLIER s  where s.CITY = 'PARIS' and s.STATUS > 20;  In other words, consider a supplier. Include it in the result if its city is Paris and its status is greater than 20.  Result:   |  | | --- | | S# | | S3 | |
|  | **Example SP-4**: Get the supplier number and name of suppliers in London.  select s.S#, s.SNAME  from SUPPLIER s  where s.CITY = “London”;  Result:   |  |  | | --- | --- | | **S#** | **SNAME** | | S1 | SMITH | | S4 | FORD | |
|  | **Example SP-5**: Get the supplier number of suppliers who supply part P2.  select DISTINCT h.S# from SHIPMENT h  where h.P# = 'P2';  Result:   |  | | --- | | **S#** | | S1 | | S2 | | S3 | | S4 | |
| **Simple existential quantifiers are not necessary**  With relational calculus, we had to provide a quantifier for any tuple variables not present in the column list. With SQL, that isn't necessary unless NOT EXISTS is needed. | **Example SP-6**: Get the supplier name of suppliers of part P2.  select s.SNAME  from SUPPLIER s, SHIPMENT h  where h.P# = 'P2' and h.S# = s.S#;  Result:   |  | | --- | | SNAME | | SMITH | | JONES | | BLAKE | | FORD |   **Example SP-7:**  Get the part name of parts supplied by supplier S2.  select p.PNAME  from Parts p, SHIPMENT h  where h.S# = ‘S2’ and p.P# = h.P#; |
| |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **SUPPLIER**   |  |  |  |  | | --- | --- | --- | --- | | **S#** | **SNAME** | **STATUS** | **CITY** | | S1 | SMITH | 20 | LONDON | | S2 | JONES | 10 | PARIS | | S3 | BLAKE | 30 | PARIS | | S4 | FORD | 20 | LONDON | | S5 | ADAMS | 30 | ATHENS |   **Parts**   |  |  |  | | --- | --- | --- | | **P#** | **PNAME** | **COLOR** | | P1 | NUT | RED | | P2 | BOLT | GREEN | | P3 | SCREW | BLUE | | P4 | SCREW | RED | | P5 | CAM | BLUE | | P6 | COG | RED | | **SHIPMENT**   |  |  |  | | --- | --- | --- | | **S#** | **P#** | **QTY** | | S1 | P1 | 300 | | S1 | P2 | 200 | | S1 | P3 | 400 | | S1 | P4 | 200 | | S1 | P5 | 100 | | S1 | P6 | 100 | | S2 | P1 | 300 | | S2 | P2 | 400 | | S3 | P2 | 200 | | S4 | P2 | 200 | | S4 | P4 | 300 | | S4 | P5 | 400 | | | **Example SP-8:**  Get the city of suppliers who supply part P2.  Could there be multiple suppliers from the same city the supply P2? Yes  //DISTINCT FOR NO DUPS  Select DISTINCT s.CITY  from SUPPLIER s, SHIPMENT h  where h.P# = ‘P2’ and h.S# = s.S#; |
| |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **SUPPLIER**   |  |  |  |  | | --- | --- | --- | --- | | **S#** | **SNAME** | **STATUS** | **CITY** | | S1 | SMITH | 20 | LONDON | | S2 | JONES | 10 | PARIS | | S3 | BLAKE | 30 | PARIS | | S4 | FORD | 20 | LONDON | | S5 | ADAMS | 30 | ATHENS |   **Parts**   |  |  |  | | --- | --- | --- | | **P#** | **PNAME** | **COLOR** | | P1 | NUT | RED | | P2 | BOLT | GREEN | | P3 | SCREW | BLUE | | P4 | SCREW | RED | | P5 | CAM | BLUE | | P6 | COG | RED | | **SHIPMENT**   |  |  |  | | --- | --- | --- | | **S#** | **P#** | **QTY** | | S1 | P1 | 300 | | S1 | P2 | 200 | | S1 | P3 | 400 | | S1 | P4 | 200 | | S1 | P5 | 100 | | S1 | P6 | 100 | | S2 | P1 | 300 | | S2 | P2 | 400 | | S3 | P2 | 200 | | S4 | P2 | 200 | | S4 | P4 | 300 | | S4 | P5 | 400 | | | **Example SP-9:**  Get the supplier number of suppliers who supply at least one red part.  select DISTINCT h.S#  from SHIPMENT h, PART p  where p.COLOR = 'RED' and p.P# = h.P#;  Result:   |  | | --- | | S# | | S1 | | S2 | | S4 | |
|  | **Example SP-10:**  Get the name of suppliers who supply at least one red part.  select DISTINCT s.SNAME  from SHIPMENT h, PART p, SUPPLIER s  where h.S# = s.S# and p.COLOR = 'RED' and p.P# = h.P#; |
|  | **Example SP-11:**  Get the S# and names of suppliers who supply at least one part supplied by supplier S2.  select DISTINCT s.S#, s.SNAME  from SHIPMENT h, SUPPLIER s, SHIPMENT hs2  where hs2.S# = 'S2'  and hs2.P# = h.P# and h.S# = s.S#;     |  |  | | --- | --- | | **S#** | **SNAME** | | S1 | SMITH | | S2 | JONES | | S3 | BLAKE | | S4 | FORD |   Note that S2 JONES is in the result. How can we eliminate S2 Jones?  select DISTINCT s.S#, s.SNAME  from SHIPMENT h, SUPPLIER s, SHIPMENT hs2  where hs2.S# = 'S2' and h.S# <> 'S2'  and hs2.P# = h.P# and h.S# = s.S#; |
| **Universal Quantification**  What does the query on the right mean?  A supplier should be included in the result if **none of its** shipments are for part P1.  **SHIPMENT**   |  |  |  | | --- | --- | --- | | **S#** | **P#** | **QUANTITY** | | S1 | P1 | 300 | | S1 | P2 | 200 | | S1 | P3 | 400 | | S1 | P4 | 200 | | S1 | P5 | 100 | | S1 | P6 | 100 | | S2 | P1 | 300 | | S2 | P2 | 400 | | S3 | P2 | 200 | | S4 | P2 | 200 | | S4 | P4 | 300 | | S4 | P5 | 400 | | **Example SP-12:**  Get the supplier names of suppliers who do **not** supply part P1.  Using relational calculus:  RANGE SUPPLIER s  RANGE SHIPMENT h  Get w (s.SNAME): ∀h (s.S# = h.S# → h.P# ≠ 'P1')  Consider a supplier. Include its name in the result if for all shipments by that supplier the P# is not P1.  Converting to SQL:  Get w (s.SNAME): ∀h (s.S# ≠ h.S# v h.P# ≠ 'P1')  Get w (s.SNAME): ~~∀h (s.S# ≠ h.S# v h.P# ≠ 'P1')  Get w (s.SNAME): ~∃h (~(s.S# ≠ h.S# v h.P# ≠ 'P1'))  Get w (s.SNAME): ~∃h (s.S# = h.S# ^ h.P# = 'P1')  select s.SNAME  from SUPPLIER s  where NOT EXISTS (  Select True  From SHIPMENT h  Where s.S# = h.S# and h.P# = 'P1');  select s.SNAME  from SUPPLIER s  where NOT EXISTS (  Select True  From SHIPMENT h  Where s.S# = h.S# and h.P# = 'P1');  And EXISTS (  Select True |
| **ALL - the result will typically have two NOT EXISTS clauses** | **Example SP-13:**  Get the names of suppliers who supply **all** of the parts.  Consider a supplier. For all the parts, there must be a shipment of the part by the supplier.  RANGE SUPPLIER s  RANGE SHIPMENT h  RANGE PART p  Get w (s.SNAME): ∀p ∃h (s.S# = h.S# ^ h.P# = p.P#)  Converting to SQL:  Get w (s.SNAME): ~~∀p ∃h (s.S# = h.S# ^ h.P# = p.P#)  Get w (s.SNAME): ~∃p ~∃h (s.S# = h.S# ^ h.P# = p.P#)  select s.SNAME  from SUPPLIER s  where NOT EXISTS (  select p.P#  from PART p  where NOT EXISTS (  select h.S#  from SHIPMENT h  where h.S# = s.S# AND h.P# = p.P#)); |
|  | **Example SP-14:**  Get the supplier numbers of suppliers who supply all the parts.  select s.S#  from SUPPLIER s  where NOT EXISTS (  select p.P#  from PART p  where NOT EXISTS (  select h.S#  from SHIPMENT h  where h.S# = s.S# AND h.P# = p.P#)); |
| **ONLY** | **Example SP-15:** Get the names of suppliers who **only** supply part P2.  Consider a supplier. Include its name in the result if for all shipments by that supplier the P# is P2.  RANGE SUPPLIER s  RANGE SHIPMENT h  Get w (s.SNAME): ∀h (s.S# = h.S# → h.P# = 'P2')  Converting to SQL:  Get w (s.SNAME): ∀h (s.S# ≠ h.S# v h.P# = 'P2')  Get w (s.SNAME): ~~∀h (s.S# ≠ h.S# v h.P# = 'P2')  Get w (s.SNAME): ~∃h (~(s.S# ≠ h.S# v h.P# = 'P2'))  Get w (s.SNAME): ~∃h (s.S# = h.S# ^ h.P# ≠ 'P2'))  select s.SNAME  from SUPPLIER s  WHERE NOT EXISTS (  Select True  From SHIPMENT h  Where s.S# = h.S# and h.P# <> 'P2'); |
| **ALL of a subset**  First decide what we want all of …  For all red parts, the supplier must have a shipment. | **Example SP-16:** Get the names of suppliers who supply **all** of the red parts.  RANGE SUPPLIER s  RANGE SHIPMENT h  RANGE PART p  Get w (s.SNAME): ∀p (p.COLOR = 'RED'  → ∃h (h.P# = p.P# ^ s.S# = h.S#))  Converting to SQL:  Get w (s.SNAME): ∀p (p.COLOR ≠ 'RED'  v ∃h (h.P# = p.P# ^ s.S# = h.S#))  Get w (s.SNAME): ~~∀p (p.COLOR ≠ 'RED'  v ∃h (h.P# = p.P# ^ s.S# = h.S#))  Get w (s.SNAME): ~∃p (~(p.COLOR ≠ 'RED'  v ∃h (h.P# = p.P# ^ s.S# = h.S#)))  Get w (s.SNAME): ~∃p (p.COLOR = 'RED'  ^ ~∃h (h.P# = p.P# ^ s.S# = h.S#))  SQL:  select s.SNAME  from SUPPLIER s  where NOT EXISTS (  select True  from Part p  where p.COLOR = ‘RED’ and NOT EXISTS (  select True  from SHIPMENT h  where h.P# = p.P# and s.S# = h.S#)); |
| **ONLY**  **Supplier Parts DB sample Data**   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **SUPPLIER**   |  |  |  |  | | --- | --- | --- | --- | | **S#** | **SNAME** | **STATUS** | **CITY** | | S1 | SMITH | 20 | LONDON | | S2 | JONES | 10 | PARIS | | S3 | BLAKE | 30 | PARIS | | S4 | FORD | 20 | LONDON | | S5 | ADAMS | 30 | ATHENS |   **Parts supplied by suppliers not in London**   |  | | --- | | **P#** | | P1 | | P2 | | P3 | | P4 | | P5 | | P6 | | **SHIPMENT**   |  |  |  | | --- | --- | --- | | **S#** | **P#** | **QTY** | | S1 | P1 | 300 | | S1 | P2 | 200 | | S1 | P3 | 400 | | S1 | P4 | 200 | | S1 | P5 | 100 | | S1 | P6 | 100 | | S2 | P1 | 300 | | S2 | P2 | 400 | | S3 | P2 | 200 | | S4 | P2 | 200 | | S4 | P4 | 300 | | S4 | P5 | 400 | | | **Example SP-17:** Get the name and part number of parts **only** supplied by suppliers in  London.  RANGE SUPPLIER s  RANGE SHIPMENT h  RANGE PART p  Get W17 (p.P#, p.PNAME): ∀h (h.P# = p.P#  → ∃s (s.S# = h.S# ^ s.CITY = 'London'))  Converting to SQL:  ∀h (h.P# != p.P# v ∃s(s.S# = h.S# ^ s.CITY = ‘London’))  ~~∀h (h.P# != p.P# v ∃s(s.S# = h.S# ^ s.CITY = ‘London’))  ~∃h ~(h.P# != p.P# v ∃s(s.S# = h.S# ^ s.CITY = ‘London’))  ∃h (h.P# != p.P# ^ ∃s(s.S# = h.S# ^ s.CITY = ‘London’))  SQL:  select p.P#, p.PNAME  from PART p  where NOT EXISTS (  select True  from SHIPMENT h  where h.P# = p.P# AND NOT EXISTS (  select True  from SUPPLIER s  where s.S# = h.S# and s.CITY = ‘London’));  Which parts are in the result?  **Result:**   |  |  | | --- | --- | | **P#** | **PNAME** | | P3 | SCREW | | P4 | SCREW | | P5 | CAM | | P6 | COG | |
| **Selection from multiple relations with multiple criteria** | **Example SP18:** For shipments with a quantity < 500 by suppliers with a status greater than 10 of a part with a weight greater than 12, get the part#.  select p.P#  from PART p, SUPPLIER s, SHIPMENT h  where h.QUANTITY < 500  and s.STATUS > 10  and p.Weight > 12  and h.P# = p.P# //joins  and h.S# = s.S#; //joins    Result:   |  | | --- | | P# | | P2 | | P3 | | P4 | | P6 | |
| **Selection from multiple relations with multiple criteria** | **Example SP19:** For shipments with a quantity < 500 by suppliers with a status greater than 10 of a part with a weight greater than 12, get the part number, part name, supplier number, and supplier name.  select p.P#, p.PNAME, s.S#, s.SNAME  from PART p, SUPPLIER s, SHIPMENT h  where h.QUANTITY < 500  and s.STATUS > 10  and p.Weight > 12  and h.P# = p.P# //joins  and h.S# = s.S#; //joins  Result:   |  |  |  |  | | --- | --- | --- | --- | | P# | PNAME | S# | SNAME | | P2 | BOLT | S1 | SMITH | | P2 | BOLT | S3 | BLAKE | | P2 | BOLT | S4 | FORD | | P3 | SCREW | S1 | SMITH | | P4 | SCREW | S1 | SMITH | | P4 | SCREW | S4 | FORD | | P6 | COG | S1 | SMITH | |
|  |  |
|  |  |

**©2020 Larry W. Clark,** UTSA CS students may make copies for their personal use