|  |  |  |
| --- | --- | --- |
| **SQL Query** | **Explanation** | **Example** |
| * Expressions use standard arithmetic operators (+, -, \*, /) on numeric values or attributes with numeric domains * [] denote optional additions * **Red text** denotes Relational Information () * **Yellow text** denote abreviation of SQL (only full version can be used) * **Black text** denotes SQL QUERY   [\_\_] DDL – Data Definition Language  [\_\_] DML – Data Management Language  [\_\_] TCL – Transactional Control Language  Domain (Data-type):  CHAR(x) – a character string of size x  INTEGER/INT(x) – integer of size x  FLOAT(x, d) – a floating point of size x with d digits after decimal point  DATE – date of format YYYY-MM-DD  DATETIME – date of format YYYY-MM-DD hh:mm:ss  BOOLEAN/BOOL – true or false (0=false, 1=true) | | |
| IC 🡪 [INITIALLY [IMMEDIATE, DEFERRED] [NON DEFERRABLE, DEFERRABLE] CONSTRAINT constraint-name]  CREATE TABLE table-name(  attribute-name: data-type [NOT NULL, DEFAULT x],  …  UNIQUE(attribute-name) IC,  PRIMARY KEY(attribute-name) IC,  FOREIGN KEY (attribute-name) REFERENCES table-name (attribute-name) IC,  ON DELETE [SET NULL, SET DEFAULT x, CASCADE, NO ACTION],  ON UPDATE [SET NULL, SET DEFAULT x, CASCADE, NO ACTION],  CHECK(qualification) IC  ); | **Creates relations:**  **UNIQUE( \_ )** : constraint ensures all elements of \_ are unique  **INITIALLY \_ \_** : specifies how constraint will be handled  **CONSTRAINT \_ \_** : defines a name \_ for a constraint \_  **PRIMARY KEY( \_ )** : constraints \_ to uniquely identify each tuple in relation  **FOREIGN KEY ( \_ ) REFERENCES \_ ( \_ )** : constraints attribute \_ to reference a key in another relation  **ON DELETE \_** : instructions on what to do when tuple is deleted  **ON UPDATE \_** : instructions on what to do when tuple is updated  **CHECK ( \_ )** : defines semantic constraints on specific attributes/tuples | CREATE TABLE StarIn(  movieID: INTEGER(10) NOT NULL,  starID: INTEGER(10) NOT NULL,  role: CHAR(100),  PRIMARY KEY(movieID, starID),  FOREIGN KEY (movieID) REFERENCES Movie (movieID),  FOREIGN KEY (starID) REFERENCES MovieStar (starID)  ); |
| ALTER TABLE table-name  ADD [CONSTRAINT] attribute-name: data-type  DROP COLUMN attribute-name [SET NULL, SET DEFAULT, CASCADE]  ALTER attribute-name: data-type  RENAME COLUMN attribute-name TO attribute-name;  MODIFY attribute-name data-type;  ; | **Alters relations:**  **ADD \_** : adds new attribute \_  **DROP COLUMN \_** : deletes attribute \_  **ALTER \_** : changes attribute \_  **RENAME COLUMN\_TO\_** : renames attribute \_ to \_  **MODIFY \_ \_** : modifies the attribute \_’s domain to \_ | ALTER TABLE Movie(  ADD favourite-colour: CHAR(100)  ); |
| INSERT INTO table-name (attribute-name, …) VALUES (x, …); | **Inserts tuples into relations** | INSERT INTO Movie VALUES(1, ‘Ran’,1985); |
| UPDATE table-name  SET attribute-name = x  WHERE qualification; | **Updates tuples’ values:** | UPDATE Movie  SET year = year+10  WHERE year<1940; |
| DESC table-name; | **shows relation’s structure:** | DESC Movie; |
| DELETE FROM table-name WHERE condition; | **Deletes tuples from relations** | DELETE FROM Movie WHERE year < 1950; |
| DROP TABLE table-name; | **Deletes relation** | DROP TABLE Movie; |
| CREATE ASSERTION assertion-name  CHECK NOT EXISTS (\*\*SQL QUERY\*\*) IC; | **Creates a Global semantic constraint:**  **\*\*SQL QUERY\*\*** : a query that retrieves any tuple that violates the wanted conditions | CREATE ASSERTION PreWW2Movies  INITIALLY DEFERRED DEFERRABLE CHECK NOT EXISTS(  SELECT \*  FROM Movie  WHERE year<1939  ); |
| CREATE DOMAIN domain-name AS data-type CHECK(qualification) IC; | **Creates a custom domain with an in-built constraint:** | CREATE DOMAIN movieYear AS INTEGER  CHECK(movieYear>1940); |
| [CREATE,REPLACE] TRIGGER trigger-name  [BEFORE,AFTER] [INSERT,DELETE,UPDATE] ON relation-name  [FOR EACH [ROW,STATEMENT]]  WHEN(EXISTS(\*\*SQL QUERY\*\*))  BEGIN  action  END;/ | **Creates a Global Trigger:**  **\_ TRIGGER \_** : defines the trigger of name \_  **\_ \_ ON \_** : defines the trigger time or \_ on condition \_ on relations \_  **FOR EACH \_** : would trigger once for every violation of \_  **WHEN(EXIST(** : defines the condition which the trigger is tripped  **BEGIN** : begins action  **ENDS;/** : ends action and activates trigger | Notify the Dean when the # of students in any major exceeds 1800 |
| [CREATE [MATERIALIZED] VIEW view-name(attribute-name, …) AS]  SELECT [DISTINCT] select-list [COUNT(), AVG(), SUM(), MAX(), MIN(), \*, AS, ISNULL(FieldName, NewVal)]  FROM from-list  [  [FULL OUTER, LEFT, RIGHT] JOIN sub-from-list ON sub-qualification  WHERE qualification [NOT, OR, AND, BETWEEN, NULL, IN(), ALL(), ANY(), EXIST()]  GROUP BY grouping-list  HAVING group-qualification  ORDER BY ordering-list [ASC, DESC]  ]  [UNION, INTERSECT, EXCEPT] | **Retrieves tuples from relations:**  **CREATE VIEW \_ AS :** Creates a virtual relation of name \_  **MATERIALIZED**: Creates a materialised view  **SELECT \_ :** Retrieves lists of attributes \_ from relation(s)  **FROM \_ :** Specifies the relation(s) in question (Multiple relations selected will cause a **cross-product** relation to occur)  **WHERE \_ :** Specification that every tuple must meet  **GROUP BY \_ :** lists of attributes \_ of which to group the tuples  **HAVING \_ :** Specification that every tuple in a group must meet  **ORDER BY \_:** orders all tuples based on \_ | CREATE VIEW starsWhoActInAllMovies AS  SELECT \*  FROM MovieStar AS MS WHERE NOT EXIST(  SELECT M.movieID  FROM Movie AS M  WHERE MovieID NOT IN(  SELECT SI.movieID  FROM StartsIn AS SI  WHERE SI.startID = MS.starID  )  ); |
| SET TRANSACTION [READ WRITE, READ ONLY, WRITE ONLY ] NAME transaction-name;  SET CONSTRAINT [ALL, constraint-name] [IMMEDIATE, DEFERRED] [NON DEFERRABLE, DEFERRABLE];  \*\*SQL QUERY\*\*  [COMMIT,ROLLBACK]; | **Creates a transaction instance:**  **SET TRANSACTION \_ NAME \_** : starts a transaction with database  **SET CONSTRAINT \_ \_** : specifies how transactions are evaluated  **COMMIT** : saves the transaction onto the database  **ROLLBACK** : undoes the previous transaction that have not been saved in database | SET TRANSACTION READ WRITE NAME test;  SET CONTRAINT ALL DEFERRED DEFERRABLE  SELECT \* FROM Movie  COMMIT; |
| SET TIMING ON; | **Configure SQL\*Plus to display query execution time:** | SET TIMING ON; |
| EXPLAIN PLAN FOR query;  SELECT PLAN\_TABLE\_OUTPUT FROM TABLE (DBMS\_XPLAN.DISPLAY); | **Explains steps performed during query execution** | EXPLAIN PLAN FOR SELECT \* FROM Movie;  SELECT PLAN\_TABLE\_OUTPUT FROM TABLE (DBMS\_XPLAN.DISPLAY); |
| **Conceptual Evaluation** | | |