SELECT:

```
SELECT col1, col2
FROM table
WHERE condition
GROUP BY cols
HAVING condition
ORDER BY col;
```

DISTINCT: *Removes duplicate results.*
BETWEEN: *Matches a value between two other values (inclusive).*
IN: *Matches a value to one of many values.*
LIKE: *Performs partial/wildcard matches.*

ALIASES:

```
SELECT studentID, mark, maxmarks, round(100 *
mark / maxmarks,2) AS grade
FROM studentMarks
WHERE grade >= 50;  -- will fail
-- needs to be:
WHERE round(100 * mark / maxmarks,2) >= 50;
```

WILDCARDS:

%: The percent wildcard specifies that any characters can appear in multiple positions.
_: The underscore wildcard specifies a single position in which any character can occur.

```
SELECT firstname, lastname
FROM customers
WHERE Upper(firstname) LIKE 'B%';
```

DATE COMPARISON:

```
SELECT * FROM employees
WHERE hiredate BETWEEN To_Date('050110',
'mmddyy') AND To_Date('053110','mmddyy')
```

sysdate: current date

LIMITING RESULTS & PAGING:

```
SELECT * FROM orders
ORDER BY orderdate
OFFSET 20 ROWS -- optional to offset
FETCH NEXT 10 ROWS ONLY;
```

Comparison and Logical Operators:

Symbol / Text	Example
=	studID = 990123456
<	salary < 5000
<=	numProducts <= 10
>	salary > 5000
>=	numProducts >= 2
<> !=	name != 'Bob'
IN and NOT IN	productID IN (12, 45, 67, 65)
ANY	productID > ANY(12,45,67,76)
ALL	salary >= ALL(2000, 3000, 4000)
BETWEEN and NOT BETWEEN	salary BETWEEN (2000 and 5000)
LIKE and NOT LIKE	name LIKE 'B%'
IS NULL and IS NOT NULL	middlename IS NULL
AND	salary => 2000 AND salary <= 5000
OR	studID = 990123456 OR studID = 9906543321

INSERT:

```
INSERT INTO <tablename> ( <fieldlist comma
separated> )
VALUES ( <Value list comma separated> );
```

Inserting Multiple Rows in One Statement:

```
INSERT ALL
    INTO <tablename> VALUES (<value list>)
    INTO <tablename> VALUES (<value list>)
```

Inserting Multiple Rows From another Table:

```
INSERT INTO <tablename> (<fieldlist>)
SELECT <fieldlist>
FROM <tablename>
WHERE <condition>  -- optional
ORDER BY <fieldlist>;  -- optional
```

UPDATE:

```
UPDATE employees SET lastname = 'Smith'
WHERE employeeId = 343; --Primary Key unique
```

DELETE:

```
DELETE FROM <tablename>
WHERE <condition> -- optional
```

DATA TYPES:

Required	Choice
Fixed width strings	char(n)
Variable width strings	varchar(n)
Variable width strings with UNICODE characters	nvarchar(n)
Integers (up to 255)	number(3)
Integers (up to 32,000)	shortinteger
Integers (over 32,000 up to 2,000,000,000)	integer
Decimals	number(p,s)
Dates	Date
Booleans	number(1)

CONSTRAINTS:

Constraint	Statement
Primary Key	PRIMARY KEY
Foreign Key	FOREIGN KEY
Required	NOT NULL
Unique	UNIQUE
Default Value	DEFAULT
Check Range	CHECK
Index	CREATE INDEX

CREATE:

```
CREATE TABLE teams (
teamID          INT     PRIMARY KEY UNIQUE,
LeagueID        INT     FOREIGN KEY,
teamName        VARCHAR(15)    NOT NULL,
maxPlayersINT   DEFAULT 0,
CONSTRAINT maxPlayer_chk CHECK (maxPlayers BETWEEN
0 AND 25));
```

DELETE OBJECT:

```
DROP TABLE teams;
```

ALTER:

Adding a New Column:

```
ALTER TABLE players
    ADD date_of_birth DATE;  -- could add
constraints here too if needed, or use ALTER to
add them after.
```

Dropping a Column:

```
ALTER TABLE players
    DROP COLUMN date_of_birth;
```

Dropping a Constraint:

```
ALTER TABLE players
    DROP CONSTRAINT player_teams_fk;
```

VIEWS:

```
CREATE VIEW vwPlayersOnTeams AS
   SELECT playerid, firstname, lastname,
p.teamid pteamid, t.teamid tteamid, teamname
   FROM players p FULL JOIN teams t
   ON p.teamid = t.teamid;
SELECT * FROM vwPlayersOnTeams;
```

JOINS:

Inner Joins:

```
SELECT
        playerID,
        firstname,
        lastname,
        teamName
FROM players INNER JOIN teams
ON players.teamID = teams.teamID;
```

Left or Right Joins:

```
SELECT
        playerID,
        firstname,
        lastname,
        teamName
FROM players LEFT OUTER JOIN teams
ON players.teamID = teams.teamID;
-- OR
FROM teams RIGHT OUTER JOIN players
ON players.teamID = teams.teamID;
```

Inverse Outer Joins:

```
SELECT teamID, teamName
FROM teams LEFT OUTER JOIN players
USING (teamID)
WHERE playerID IS NULL;
```

Full Outer Joins:

```
SELECT playerID, firstname, lastname, teamName
FROM teams FULL OUTER JOIN players
USING (teamID)
```

Inverse Full Joins:

```
SELECT playerID, firstname, lastname, teamName
FROM teams FULL OUTER JOIN players
USING (teamID)
WHERE teamID IS NULL;
```

TRANSACTION:

Auto-Commit:

```
SET AUTOCOMMIT ON/OFF
```

Rollback:

```
COMMIT;  -- force starts a new transaction
INSERT INTO players (playerID, firstname)
    VALUES (1667, 'George');
SELECT * FROM players WHERE playerID = 1667;
-- there will be one record shown
ROLLBACK;
SELECT * FROM players WHERE playerID=1667;   --
no records shown
```

Savepoints:

```
COMMIT;  -- force starts a new transaction
INSERT INTO players (playerID, firstname)
    VALUES (1667, 'George');
SAVEPOINT A;
SELECT * FROM players WHERE playerID = 1667 OR
playerID = 1668
INSERT INTO players (playerID, firstname)
    VALUES (1668, 'Henry');
SELECT * FROM players WHERE playerID = 1667 OR
playerID = 1668;
ROLLBACK TO A;
SELECT * FROM players WHERE playerID = 1667 OR
playerID = 1668;
```

The Form of a Transaction with Error Checking:

```
BEGIN
   INSERT INTO players (playerID, firstname)
       VALUES (1667, 'George');
   INSERT INTO players (playerID, firstname)
       VALUES (1668, 'Henry');
   COMMIT;
   DBMS_OUTPUT.PUT_LINE('Transaction
   Successful!');
EXCEPTION
   ROLLBACK;
   DBMS_OUTPUT.PUT_LINE('Transaction Failed,
   rolled back');
END;
```

Database Relationship Diagram: