1 Creating Tables

```
CREATE TABLE xyz (
    fieldname   type            constraints,
    name        VARCHAR2(20)    NOT NULL,
    Age         NUMBER(2)           ,
    Id          CHAR(4)         PRIMARY KEY
)
```

1.1 datatypes

- CHAR(n)
- VARCHAR(n)
- INTEGER
- NUMERIC(s,p), NUMBER(s,p)
- DATE
- And many others, may vary per DBMS

1.2 Constraints

- NOT NULL
- UNIQUE
- CHECK
- DEFAULT
- PRIMARY KEY
- FOREIGN KEY/REFERENCES
- per column or per table

```
CREATE TABLE State (
    Code CHAR(2) PRIMARY KEY,
    Name VARCHAR(20) UNIQUE NOT NULL
);

CREATE TABLE City (
    Name VARCHAR(20),
    State CHAR(2),
    Population NUMERIC(8,0) DEFAULT 20000,
    Income_Per_Cap NUMERIC(6,2) NOT NULL,
    CONSTRAINT city_pk PRIMARY KEY (name,state),
    CONSTRAINT city_state_fk FOREIGN KEY
         (State) REFERENCES State(Code)
);
```

2 Add/Delete/Modify data

2.1 Insert

```
INSERT INTO xyz(name,id) VALUES ('Juan',1111)
```

2.2 Delete

```
DELETE FROM xyz
```

or

```
DELETE FROM xyz WHERE ...
```

2.3 Modify data (UPDATE)

```
UPDATE xyz SET age=21, name='Juana'
```

or

```
UPDATE xyz SET age=8, name='Juan' WHERE id=1111
```

3 Other Commands

- ALTER TABLE
- CREATE INDEX
- CREATE TABLE FROM ...
- DROP TABLE
- BEGIN Transaction / END Transaction

4 Select

```
SELECT field(s)
FROM table(s)
WHERE condition(s)
```

Example:

```
SELECT name,age
FROM xyz
WHERE age>21 AND name LIKE 'J\%';
```

- DISTINCT | ALL
- Can use expressions rather than plain fields
  - +,-, ...
  - || (string concatenation)
  - CASE expressions
  - Functions (vary per DB)
- Conditions
  - Can combine with AND, OR, NOT
  - Can put parenthesis around expressions
  - =, <>, >= ...
  - IS NULL, IS NOT NULL
  - IN (), NOT IN ()
  - >=ANY, <=ALL, ...
  - x BETWEEN y AND z
  - LIKE '%a_'
- Aliasing (renaming)
  - After field name, renames field. SELECT x AS y, ...
  - After table (can use AS) FROM Bar AS b ...

4.1 ORDER BY

Can add an ORDER BY clause at the end of a select, to get results sorted.

4.2 aggregates

- AVG()
- MIN()
- MAX()
- SUM()
- COUNT()

4.3 GROUP BY

Can group aggregate functions, by adding a GROUP BY clause

4.4 HAVING

Can 'select' rows AFTER grouping, based on results of aggregate functions, with a HAVING clause.

```
SELECT dno, avg(salary), max(salary), min(salary)

FROM salaries

WHERE ...

GROUP BY dno

HAVING max(salary)>2*min(salary)
```

4.5 Joins

- Can SELECT FROM several tables, and use join conditions in WHERE
- Can use explicit joins (goes in FROM, in place of table name, better to name by using AS)
  - JOIN

    ```
    SELECT S.name
    FROM Student S JOIN Major M ON S.Major=M.Code
    WHERE M.M_Name='Computer Science'
    ```
  - NATURAL JOIN

    ```
    SELECT *
    FROM Student NATURAL JOIN Degree
    ```
  - LEFT | RIGHT | FULL OUTER JOIN

    ```
    SELECT *
    FROM Student LEFT OUTER JOIN Major ON major=c
    ```
  - Can use ON to specify conditions
  - Can use USING to specify fields for partial natural join

4.6 Nesting

Can use another select query in:

- IN clauses
- Instead of tables in FROM
- EXISTS
- >=ALL, <=ANY ...

4.7 UNION, INTERSECT, MINUS

```
SELECT ...
UNION
SELECT ...
```