FIT2094 Exam Notes

1. Introduction to Databases

Why databases?

Some issues faced by file system data processing:

- Lengthy development times
- Difficulty of getting quick answers
- Complex system administration
- Lack of security and limited data sharing
- Extensive programming

Databases are the best way to store and manage data - makes data persistent and shareable in a safe way. Specialised structures allow computer-based systems to store, manage and retrieve data very quickly.

Database system consists of logically related data in a single logical repository. DBMS eliminated problems of data inconsistency, data anomalies, structural dependence. DBMSs also store data structures AND the relationships between the structures and the access paths to those structures

Data Redundancy

Data redundancy occurs when the same data is stored unnecessarily at different places. It results in:

- Poor data security
- Data inconsistency
- Data-entry errors
- Data integrity problems

2. Relational Model

Definitions

Table	Persistent representation of a logical relation. Contains a group of related entity occurrences (an entity set)
Relational database	Collection of normalised relations
Domain	The column's range of permissible values
Determination	State in which knowing the value of one attribute makes it possible to determine the value of another

Functional dependence	The value of one or more attributes determines the value of one or more attributes
Determinant	An attribute whose value determines another
Dependent	Attribute whose value is determined by the other
Full functional dependence	When an entire collection of attributes in the determinant is necessary for the relationship

Characteristics of a relational table:

1. A table is perceived as a two-dimensional structure composed of rows and columns
2. Each table row (tuple) represents a single entity occurrence within the entity set
3. Each table column represents an attribute, and each column has a distinct name
4. Each intersection of a row and a column represents a single data value
5. All values in a column must conform to the same data format
6. Each column has a specific range of value known as the attribute domain
7. The order of the rows and columns is immaterial to the DBMS
8. Each table must have an attribute or combination of attributes that uniquely defines each row

Relation Properties:

1. No duplicate tables
2. Tuples are unordered within a relation
3. No ordering of attributes within a tuple

Definitions for Keys

Key	One or more attributes that determine other attributes
Primary key	A candidate key selected to uniquely identify all other attribute values in any given row
Composite key	Key composed of more than one attribute
Key attribute	An attribute that is part of a key
Super key	Attribute or group of attributes that can uniquely identify any row in the table (can contain unnecessary attributes)
Candidate key	A minimal super key (does not contain a subset that is a superkey)
Foreign key	Attribute or group of attributes in one table whose values must either match the PK in another table or be null (PK from another table has been placed to create a common attribute)
Secondary key	Key strictly used for data retrieval purposes (e.g. cust name instead of cust id). Does not always yield unique outcomes

Integrity

Entity Integrity

Entity integrity is a condition in which each row (entity instance) in the table has its own unique identity (needs to be ensured by the primary key). Requirements for ensuring entity integrity:

1. All of the values in the PK must be unique
2. No key attribute in the PK can contain a null

Referential Integrity

Referential integrity is a condition in which every reference to an entity instance by another entity instance is value. Requirements for ensuring referential integrity:

1. FK used to ensure referential integrity
2. Every FK entry must be null or a valid value in the PK of the related table

Integrity rules

Entity Integrity	Description
Requirement	All primary key entries are unique, no part of a primary key may be null
Purpose	Each row will have a unique identity and foreign key values can properly reference primary key values
Example	No invoice can have a duplicate number, nor can it be null; In Short, all invoices are uniquely identified by their invoice numbers

Referential Integrity	Description
Requirement	Foreign key may have either a null entry, as long as it is not part of the table's primary key OR an entry that matches the primary key value in a table to which it is related (every non-null foreign key value MUST reference an existing primary key value)
Purpose	Possible for an attribute not to have a corresponding value → Will be impossible to have an invalid entry, the enforcement of the referential integrity rule makes it impossible to delete a row in one table whose primary key has mandatory matching foreign key values in another table

Example	A customer might not yet have an assigned sales representative (number), but it will be impossible to have an invalid sales representative (number)

Note: flags are used to avoid nulls to indicate absence of a value

Relational Algebra

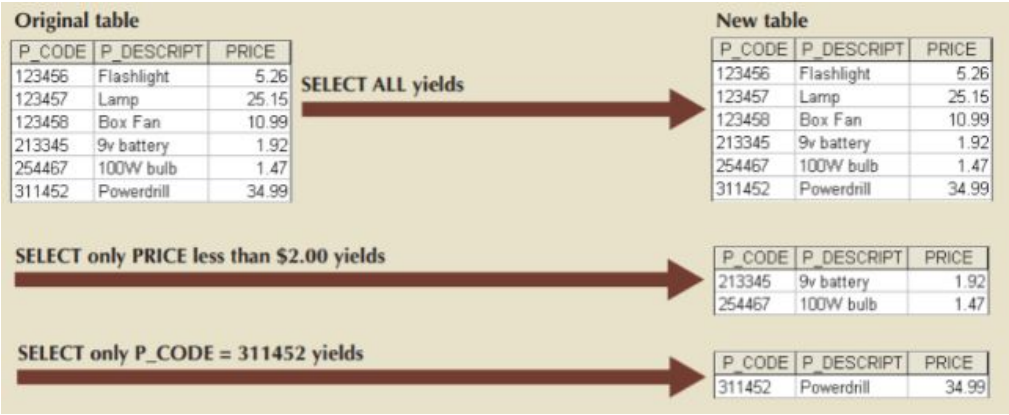
There are 8 basic operations, Selection, Projection, Cartesian Join, Cartesian Product, Union, Intersection, Difference and Division. These can be represented using the symbols:

Selection	σ	Cartesian product	$\times$
Projection	π	Join	$\bowtie$
Renaming	ρ	Logical AND	$\wedge$
Union	$\cup$	Logical OR	$\vee$
Intersection	$\cap$	Logical NOT	$\sim$
Difference	$-$		

SELECT

Yields values for all rows found that satisfy the given condition

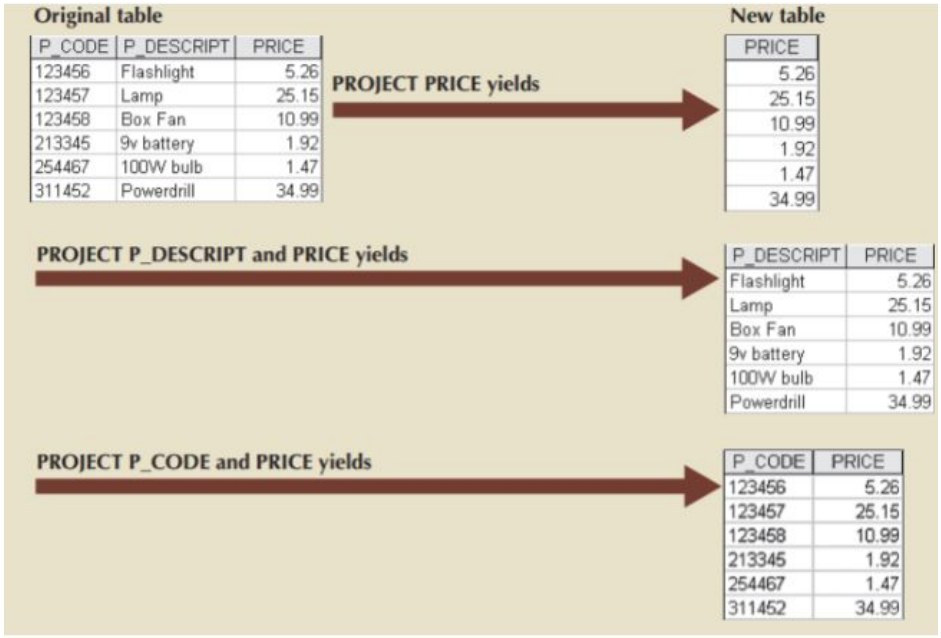
- Can list all of the rows OR can yield rows that match the criteria
- Yields a horizontal subset of a table
- **All attributes will be returned**



PROJECT

Yields all values for selected attributes

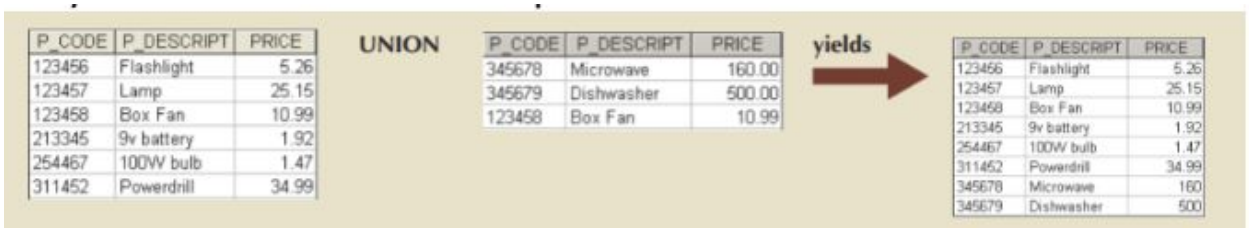
- Will return the attributes requested (in the order in which they are requested)
- Yields a vertical subset of a table
- Will not limit the rows returned so all rows of the specified attribute will be included



UNION

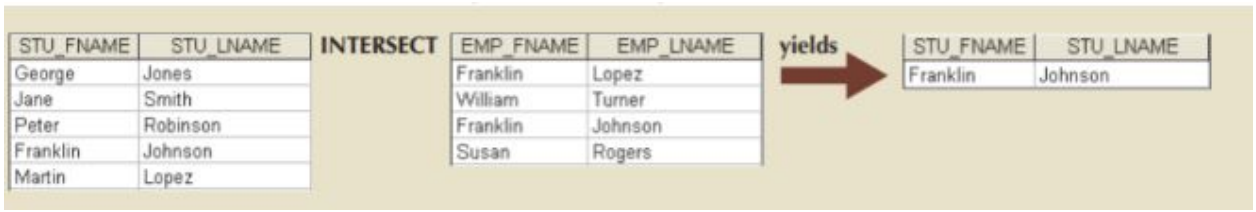
Combines all rows from two tables, excluding duplicate rows

- Columns and domains must be compatible
- When two or more tables share the same number of columns AND when their corresponding columns share the same or compatible domains , they are said to be **union-compatible**



INTERSECT

Yields only the rows that appear in both tables. Must be union compatible to yield valid results



DIFFERENCE

Yields all rows in one table that are not found in other tables. Tables must be union compatible to yield valid results

STU_FNAME	STU_LNAME
George	Jones
Jane	Smith
Peter	Robinson
Franklin	Johnson
Martin	Lopez

DIFFERENCE

EMP_FNAME	EMP_LNAME
Franklin	Lopez
William	Turner
Franklin	Johnson
Susan	Rogers

yields

STU_FNAME	STU_LNAME
George	Jones
Jane	Smith
Peter	Robinson
Martin	Lopez

PRODUCT

Yields all possible pairs of rows from two tables (cartesian product). Output has n x m rows

P_CODE	P_DESCRIPT	PRICE
123456	Flashlight	5.26
123457	Lamp	25.15
123458	Box Fan	10.99
213345	9v battery	1.92
254467	100W bulb	1.47
311452	Powerdrill	34.99

PRODUCT

STORE	AISLE	SHELF
23	W	5
24	K	9
25	Z	6

yields

P_CODE	P_DESCRIPT	PRICE	STORE	AISLE	SHELF
123456	Flashlight	5.26	23	W	5
123456	Flashlight	5.26	24	K	9
123456	Flashlight	5.26	25	Z	6
123457	Lamp	25.15	23	W	5
123457	Lamp	25.15	24	K	9
123457	Lamp	25.15	25	Z	6
123458	Box Fan	10.99	23	W	5
123458	Box Fan	10.99	24	K	9
123458	Box Fan	10.99	25	Z	6
213345	9v battery	1.92	23	W	5
213345	9v battery	1.92	24	K	9
213345	9v battery	1.92	25	Z	6
311452	Powerdrill	34.99	23	W	5
311452	Powerdrill	34.99	24	K	9
311452	Powerdrill	34.99	25	Z	6
254467	100W bulb	1.47	23	W	5
254467	100W bulb	1.47	24	K	9
254467	100W bulb	1.47	25	Z	6

Examples

1. HOTEL: PROJECT NAME and ADDRESS
   a. $\textbf{AnswerA} = \pi_{\text{name, address}}\ \textbf{HOTEL}$
2. ROOM: SELECT only PRICE less than $50
   a. $\textbf{AnswerB} = \sigma_{\text{type='single' and price < 50}}\ \textbf{ROOM}$
3. GUEST: PROJECT NAME and ADDRESS
   a. $\textbf{AnswerC} = \pi_{\text{name, address}}\ \textbf{GUEST}$
4. PROJECT PRICE and TYPE from SELECT HOTEL-NO = (PROJECT HOTEL_NO from SELECT NAME = "Grosvenor Hotel")

$$\textbf{GrosvenorNo} = \pi_{\text{hotel-no}}\ (\sigma_{\text{name = 'Grosvenor'}}\ \textbf{HOTEL})$$
$$\textbf{AnswerD} = \pi_{\text{price, type}}\ (\textbf{GrosvenorNo} \bowtie \textbf{ROOM})$$

   a. **or**

$$\textbf{AnswerD} = \pi_{\text{price, type}}\ ((\pi_{\text{hotel-no}}\ (\sigma_{\text{name = 'Grosvenor'}}\ \textbf{HOTEL})) \bowtie \textbf{ROOM})$$

3. DB Design I: Conceptual

Definitions- Entities & Attributes

Entity	Object of interest to the end user
Attribute	Characteristics of entities
Required attribute	Attribute that must have a value
Optional attribute	Attribute that does not require a value
Identifiers (PK)	One or more attributes that uniquely identify each entity instance
Composite identifier	A PK composed of more than 1 attribute
Composite attribute	Can be subdivided into additional attributes
Simple attribute	Cannot be further subdivided into additional attributes
Multivalued attribute	Can have many values (e.g. uni degrees)
Single-valued attribute	Can only have a single value (e.g. social security no.)
Derived attribute	Does not physically exist within the entity, it is derived through an algorithm

Storing Derived Attributes - Pros & Cons

Pros	**Cons**
Saves CPU processing cycles	Requires constant maintenance to ensure value is current
Saves data access time	Needs storage space (resources required)
Data value is readily available	
Can be used to keep track of historical data	

Definitions- Relationships

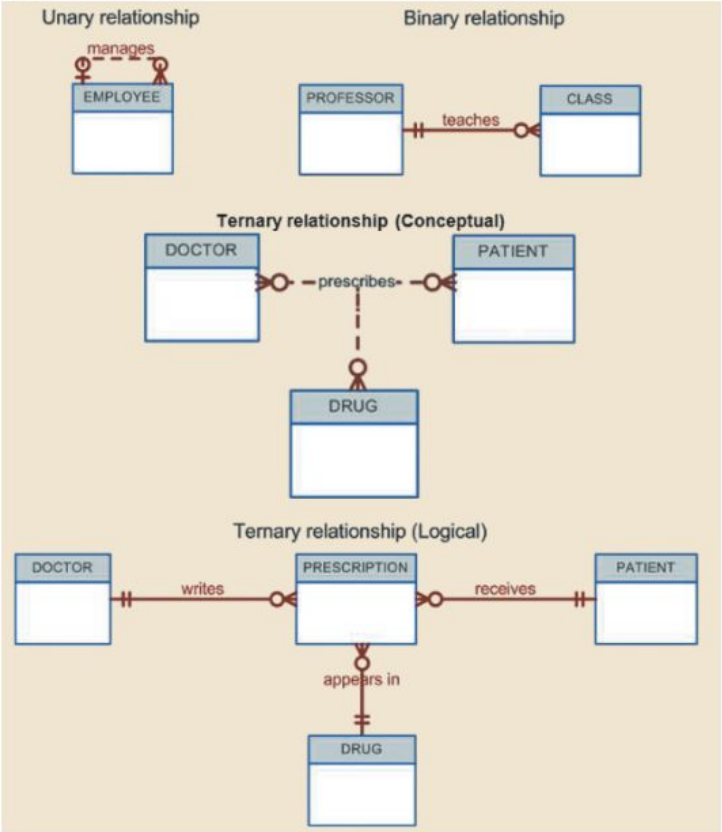
Relationship	Association between entities
Participants	Term for entities that must participate in a relationship
Connectivity	Classification of relationship: 1:1, 1:M, M:N

Cardinality	Property that assigns a specific value for connectivity. Expresses the range of allowed entity occurrences, (x, y) where x is the min number, and y is the max
Existence-dependent	Property of entity whose existence depends on one or more other entities (has mandatory FK)
Existence-independent	Can exist apart from all of its related entities
Weak (non-identifying) relationship	PK does not contain PK of parent entity
Strong (identifying) relationship	Existence dependent entities, PK contains the PK of parent
Weak entity	Existence dependent, inherits PK of parent
Strong entity	Existence independent
Optional participation	Does not require corresponding entity occurrence
Mandatory participation	One entity occurrence must have a corresponding entity occurrence in another entity
Relationship degree	Number of entities associated with a relationship

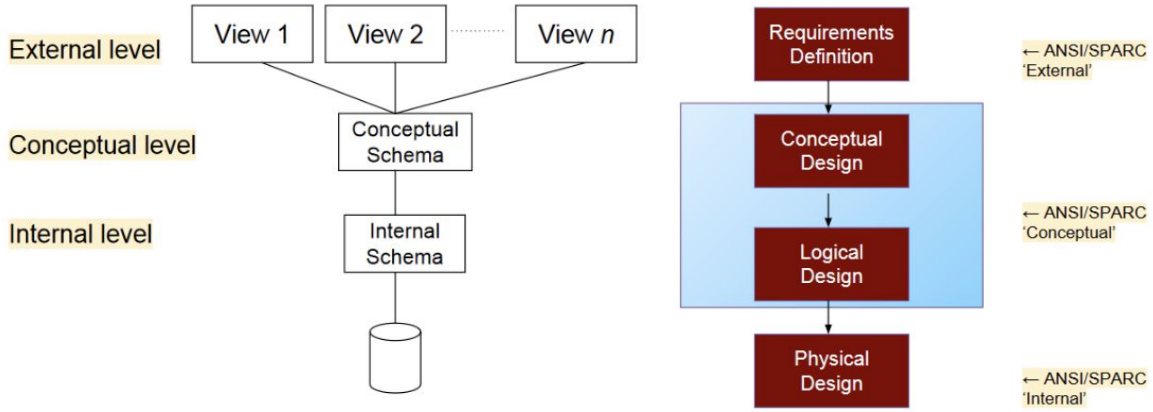
Relationship Participation - Notation

SYMBOL	CARDINALITY	COMMENT
	(0,N)	Zero or many; the "many" side is optional.
	(1,N)	One or many; the "many" side is mandatory.
	(1,1)	One and only one; the "1" side is mandatory.
	(0,1)	Zero or one; the "1" side is optional.

Relationship Degree - Notation



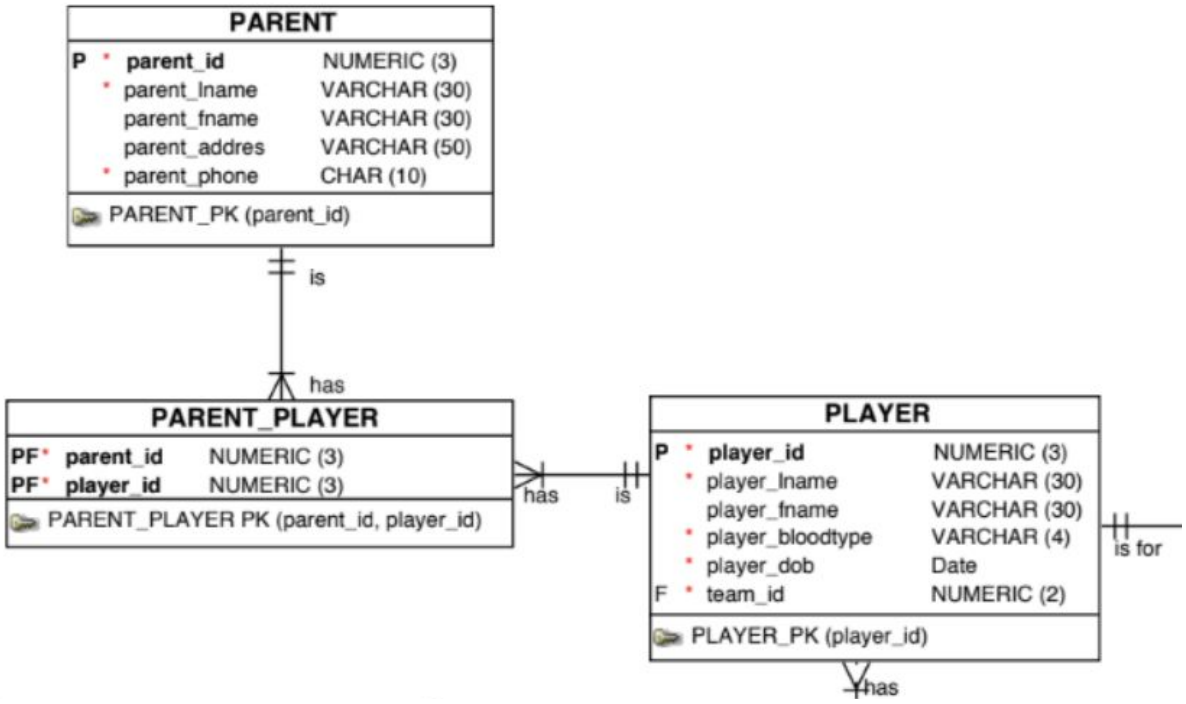
The DB Design Life Cycle



4. DB Design II: Logical

Logical design is the stage in the design phase that targets a particular DB model (e.g. relational, object-oriented). It is independent to any implementation details specific to any particular DBMS package.

Example of a logical level design (section):



5. Normalisation

Why Normalisation

Normalisation is a process that assigns attributes to entities so that data redundancies are reduced/eliminated. There are 2 main cases where we might wish to use normalisation:

1. New DB Design: structure can be improved with normalisation; reduces dependencies, redundant data and also helps in avoiding insertion, deletion and update anomalies
2. Modifying an existing structure (e.g. spreadsheets): improve and create appropriate DB design

Summary of Normal Form Characteristics

First Normal Form (1NF)	Table format, no repeating groups, PK identified
Second Normal Form (1NF)	1NF and no partial dependencies
Third Normal Form (1NF)	2NF and no transitive dependencies

Boyce-Codd Normal Form (BCNF)	3NF and every determinant is a candidate key
Fourth Normal Form (4NF)	3NF and no independent multivalued dependencies

Definitions

Partial dependency	A condition in which an attribute is dependent on only a potion (subset) of the PK
Transitive dependency	A condition in which an attribute is dependent on another attribute not in the PK
Repeating group	When, in a relation, a characteristic describing a group of multiple entries of the same type can exist for a single key attribute occurrence (has nulls)
Surrogate key	Artificial PK introduced with the purpose of simplifying the assignment of PKs to tables

Progressing Through the Stages

1NF

1. Eliminate repeating groups (eliminates nulls)
2. Identify the PK
3. Identify all dependencies and map these on the dependency diagram

2NF

1. Make a new table for each partial dependency, where the determinant is the PK
2. Reassign corresponding dependent attributes (move dependents to new table and delete from original table)

3NF

1. Make a new table for each transitive dependency, where the determinant is the PK
2. Reassign corresponding dependent attributes (move dependents to new table and delete from original table)

Improving the Design

The design can be improved after normalisation in several ways:

- Introduce a surrogate key
- Improve naming convention: e.g. JOB_CHG_HR associates the attribute with the JOB table
- Refine attribute atomicity: e.g. EMP_NAME can be broken into EMP_FNAME and EMP_LNAME
- Identify new attributes and relationships
- Maintain historical accuracy: e.g. ASSIGN_CHG_HR instead of JOB_CHG_HR

- Evaluate derived attributes, as they may create transitive dependencies

6. Creating and Populating DB (SQL Part 0)

Definitions

Schema	Logical groupings of database objects, such as tables, indexes, views, queries that are related to each other (one DB can hold multiple schemas)

Intro to SQL

SQL is a:

- Data definition language (DDL): create database objects (tables, indexes, views) and commands to define access right to those database objects
- Data manipulation language (DML): insert, update, delete and retrieve data from database tables

List of DD and DM commands included in 'extra' section.

Data Types in SQL

DATA TYPE	FORMAT	COMMENTS
Numeric	NUMBER(L,D) or NUMERIC(L,D)	The declaration NUMBER(7,2) or NUMERIC(7,2) indicates that numbers will be stored with two decimal places and may be up to seven digits long, including the sign and the decimal place (for example, 12.32 or –134.99).
	INTEGER	May be abbreviated as INT. Integers are (whole) counting numbers, so they cannot be used if you want to store numbers that require decimal places.
	SMALLINT	Like INTEGER but limited to integer values up to six digits. If your integer values are relatively small, use SMALLINT instead of INT.
	DECIMAL(L,D)	Like the NUMBER specification, but the storage length is a *minimum* specification. That is, greater lengths are acceptable, but smaller ones are not. DECIMAL(9,2), DECIMAL(9), and DECIMAL are all acceptable.
Character	CHAR(L)	Fixed-length character data for up to 255 characters. If you store strings that are not as long as the CHAR parameter value, the remaining spaces are left unused. Therefore, if you specify CHAR(25), strings such as *Smith* and *Katzenjammer* are each stored as 25 characters. However, a U.S. area code is always three digits long, so CHAR(3) would be appropriate if you wanted to store such codes.
	VARCHAR(L) or VARCHAR2(L)	Variable-length character data. The designation VARCHAR2(25) or VARCHAR(25) will let you store characters up to 25 characters long. However, unlike CHAR, VARCHAR will not leave unused spaces. Oracle automatically converts VARCHAR to VARCHAR2.
Date	DATE	Stores dates in the Julian date format.

- Oracle internal attributes include:
  - `sysdate`: current date/time
  - `systimestamp`: current date/time as a timestamp
  - `user`: current logged in user

Additional type examples: TIME, TIMESTAMP, REAL, DOUBLE, FLOAT

CREATE TABLE command

Syntax and example

```
CREATE TABLE tablename (
        column1         data type       [constraint] [,
        column2         data type       [constraint] ] [,
        PRIMARY KEY     (column1        [, column2]) ] [,
        FOREIGN KEY     (column1        [, column2]) REFERENCES tablename] [,
        CONSTRAINT      constraint ] );
```

```
CREATE TABLE VENDOR (
V_CODE          INTEGER         NOT NULL    UNIQUE,
V_NAME          VARCHAR(35)     NOT NULL,
V_CONTACT       VARCHAR(25)     NOT NULL,
V_AREACODE      CHAR(3)         NOT NULL,
V_PHONE         CHAR(8)         NOT NULL,
V_STATE         CHAR(2)         NOT NULL,
V_ORDER         CHAR(1)         NOT NULL,
PRIMARY KEY (V_CODE));
```

INSERT command

Syntax:

```
INSERT INTO tablename VALUES (value1, value2, …, valuen)
```

Example:

```
INSERT INTO VENDOR
    VALUES (21225,'Bryson, Inc.','Smithson','615','223-3234','TN','Y');
INSERT INTO VENDOR
    VALUES (21226,'Superloo, Inc.','Flushing','904','215-8995','FL','N');
```

Example (inserting only for required attributes):

```
INSERT INTO PRODUCT(P_CODE, P_DESCRIPT) VALUES ('BRT-345','Titanium
drill bit');
```

COMMIT/ROLLBACK commands

COMMIT	Used to save all changes
ROLLBACK	Used to undo the work performed by the current transaction. Undoes any changes since the last COMMIT. Note: only affects DM commands that add, modify, delete table rows

SELECT command

Syntax:

```
SELECT      columnlist      FROM      tablename
```

Example:

```
SELECT * FROM PRODUCT;

SELECT  P_CODE, P_DESCRIPT, P_INDATE, P_QOH, P_MIN, P_PRICE,
        P_DISCOUNT, V_CODE
FROM    PRODUCT;
```

ALTER TABLE command

Alter table is used to change the table structure, making changes such as:

- ADD to add a column
- MODIFY to change column characteristics
- DROP to delete a column from a table

```
ALTER TABLE tablename
    {ADD | MODIFY} ( columnname datatype [ {ADD | MODIFY}
    columnname datatype] );
```

Can also be used to add table constraints

```
ALTER TABLE tablename
    ADD constraint [ ADD constraint ];
```

Can remove a column or table constraint

```
ALTER TABLE tablename
    DROP {PRIMARY KEY | COLUMN columnname | CONSTRAIN
    constraintname };
```

Sequences

Sequences are independent objects in the database that generate a numeric value that can be assigned to any column in any table.

Syntax:

```
CREATE SEQUENCE name [START WITH n] [INCREMENT BY n]
[CACHE | NOCACHE]
```

where

- *name* is the name of the sequence.
- *n* is an integer value that can be positive or negative.
- *START WITH* specifies the initial sequence value. (The default value is 1.)
- *INCREMENT BY* determines the value by which the sequence is incremented. (The default increment value is 1. The sequence increment can be positive or negative to enable you to create ascending or descending sequences.)
- The *CACHE* or *NOCACHE/NO CACHE* clause indicates whether the DBMS will preallocate sequence numbers in memory. Oracle uses *NOCACHE* as one word and preallocates 20 values by default. SQL Server uses *NO CACHE* as two words. If a cache size is not specified in SQL Server, then the DBMS will determine a default cache size that is not guaranteed to be consistent across different databases.

- Use NEXTVAL to retrieve the net available value from a sequence

- Use CURRVAL to retrieve the current value of a sequence

Example:

```
INSERT  INTO  INVOICE  VALUES  (INV_NUMBER_SEQ.NEXTVAL,
                                20010, SYSDATE);
INSERT  INTO  LINE     VALUES  (INV_NUMBER_SEQ.CURRVAL,
                                1,'13-Q2/P2', 1, 14.99);
INSERT  INTO  LINE     VALUES  (INV_NUMBER_SEQ.CURRVAL,
                                2,'23109-HB', 1, 9.95);
COMMIT;
```

Drop sequences:

```
DROP SEQUENCE CUS_CODE_SEQ;
DROP SEQUENCE INV_NUMBER_SEQ;
```

Note: this only delete the sequence object, not the entered values in tables

7. SQL Part I

SELECT WHERE

Syntax

```
SELECT    columnlist
FROM      tablelist
[WHERE    conditionlist ];
```

Example

```
SELECT    P_DESCRIPT, P_INDATE, P_PRICE, V_CODE
FROM      PRODUCT
WHERE     V_CODE = 21344;
```

Note: null values not included

We can also use computed columns and aliases:

```
SELECT    P_CODE, P_INDATE, SYSDATE - 90 AS CUTDATE
FROM      PRODUCT
WHERE     P_INDATE <= SYSDATE - 90;
```

Additional Operators

AND, OR, NOT

Examples:

```
SELECT      P_DESCRIPT, P_INDATE, P_PRICE, V_CODE
FROM        PRODUCT
WHERE       P_PRICE < 50
AND         P_INDATE > '15-Jan-2016';
```

```
SELECT      P_DESCRIPT, P_INDATE, P_PRICE, V_CODE
FROM        PRODUCT
WHERE       V_CODE = 21344 OR V_CODE = 24288;
```

```
SELECT      *
FROM        PRODUCT
WHERE       NOT (V_CODE = 21344);
```

BETWEEN

```
SELECT      *
FROM        PRODUCT
WHERE       P_PRICE BETWEEN 50.00 AND 100.00;
```

NULL

```
SELECT      P_CODE, P_DESCRIPT, V_CODE
FROM        PRODUCT
WHERE       V_CODE IS NULL;
```

LIKE (Wildcards)

We can use the % and _ wildcard characters to make matches where the entire string is not known.

- %: any and all following or preceding characters are eligible

  'J%' includes Johnson, Jones, Jernigan, July, and J-231Q.

  'Jo%' includes Johnson and Jones.

  '%n' includes Johnson and Jernigan.

- _: Any one character can be substituted for the underscore

  '_23-456-6789' includes 123-456-6789, 223-456-6789, and 323-456-6789.

  '_23-_56-678_' includes 123-156-6781, 123-256-6782, and 823-956-6788.

  '_o_es' includes Jones, Cones, Cokes, totes, and roles.

Examples:

```
SELECT      V_NAME, V_CONTACT, V_AREACODE, V_PHONE
FROM        VENDOR
WHERE       V_CONTACT LIKE 'Smith%';
```

```
SELECT      V_NAME, V_CONTACT, V_AREACODE, V_PHONE
FROM        VENDOR
WHERE       V_CONTACT NOT LIKE 'Smith%';
```

IN

The IN operator is often used to more easily handle queries that use OR

```
SELECT      *
FROM        PRODUCT
WHERE       V_CODE = 21344
OR          V_CODE = 24288;
```

can be handled more efficiently with:

```
SELECT      *
FROM        PRODUCT
WHERE       V_CODE IN (21344, 24288);
```

Example (want to list for only vendors who provide products):

```
SELECT      V_CODE, V_NAME
FROM        VENDOR
WHERE       V_CODE IN (SELECT V_CODE FROM PRODUCT);
```

EXISTS

The EXISTS operator is used when there is a requirement to execute a command based on the result of another query

Example (only list vendors if there are products to order):

```
SELECT      *
FROM        VENDOR
WHERE       EXISTS (SELECT * FROM PRODUCT WHERE P_QOH <= P_MIN);
```

Additional SELECT Query Keywords

ORDER BY

Syntax:

```
SELECT      columnlist
FROM        tablelist
[WHERE      conditionlist ]
[ORDER BY   columnlist [ASC | DESC] ];
```

Example:

```
SELECT      P_CODE, P_DESCRIPT, P_INDATE, P_PRICE
FROM        PRODUCT
ORDER BY    P_PRICE DESC;
```

Note: defaults to ascending

DISTINCT

Example:

```
SELECT      DISTINCT V_CODE
FROM        PRODUCT;
```

JOIN ON

Syntax:

SELECT *column-list* FROM *table1* JOIN *table2* ON *join-condition*

Example:

```
SELECT      E.EMP_MGR, M.EMP_LNAME, E.EMP_NUM, E.EMP_LNAME
FROM        EMP E JOIN EMP M ON E.EMP_MGR = M.EMP_NUM
ORDER BY    E.EMP_MGR;
```

8. Update, Delete, Transaction Management

UPDATE

Allows attribute values to be changed in one or more rows. If no WHERE condition is specified, all rows in the table will be updated.

Syntax:

```
UPDATE      tablename
SET         columnname = expression [, columnname = expression]
[WHERE      conditionlist ];
```

Example:

```
UPDATE      PRODUCT
SET         P_INDATE = '18-JAN-2016'
WHERE       P_CODE = '13-Q2/P2';
```

```
UPDATE      PRODUCT
SET         P_INDATE = '18-JAN-2016', P_PRICE = 17.99, P_MIN = 10
WHERE       P_CODE = '13-Q2/P2';
```

DELETE

Allows data rows to be deleted from a table. Again if no WHERE specified, all rows will be affected.

Syntax:

```
DELETE FROM     tablename
[WHERE          conditionlist ];
```

Additional Data Definition Commands

CREATE TABLE

Selecting columns from an existing table

```
INSERT INTO     target_tablename[(target_columnlist)]
SELECT          source_columnlist
FROM            source_tablename;
```

Example:

```
CREATE TABLE PART(
PART_CODE                CHAR(8),
PART_DESCRIPT            CHAR(35),
PART_PRICE               DECIMAL(8,2),
V_CODE                   INTEGER,
PRIMARY KEY (PART_CODE));

INSERT INTO PART   (PART_CODE, PART_DESCRIPT, PART_PRICE, V_CODE)
SELECT             P_CODE, P_DESCRIPT, P_PRICE, V_CODE FROM
                   PRODUCT;
```

Alternatively, we can create a new table based on the selected columns and rows of an existing table:

```
CREATE TABLE PART AS
SELECT      P_CODE AS PART_CODE, P_DESCRIPT AS PART_DESCRIPT,
            P_PRICE AS PART_PRICE, V_CODE
FROM        PRODUCT;
```

ALTER TABLE

This command allows us to include integrity rules in a table:

- Defining PK:

```
ALTER TABLE     PART
   ADD          PRIMARY KEY (PART_CODE);
```

- Defining FK

```
ALTER TABLE     PART
   ADD          FOREIGN KEY (V_CODE) REFERENCES VENDOR;
```

- Defining both:

```
ALTER TABLE     LINE
   ADD          PRIMARY KEY (INV_NUMBER, LINE_NUMBER)
   ADD          FOREIGN KEY (INV_NUMBER) REFERENCES INVOICE
   ADD          FOREIGN KEY (P_CODE) REFERENCES PRODUCT;
```

DROP TABLE

Drop table allows for deleting a table from the database. Expressed as:

```
DROP TABLE PART;
```

Note: can only drop table if it is NOT on the one side of the relationship

Definitions

Transaction	Logical unit of work that must be entirely completed or entirely aborted
Consistent DB state	One in which all data integrity constraints are satisfied
Database request	E.g. transaction has 2 UPDATEs and 1 INSERT, giving 3 BD requests

Transaction Properties

A transaction must have the following properties:

1. Atomicity
2. Consistency
3. Isolation
4. Durability

9. SQL Part II (Aggregate Functions)

Aggregate Functions

COUNT

Returns the number of non-nulls in the given column. Example:

- COUNT(V_CODE), COUNT(P_CODE)
- COUNT(*) returns the number of rows (including nulls)

MAX & MIN

Finds the highest and lowest of a column. For example:

```
SELECT      P_CODE, P_DESCRIPT, P_PRICE
FROM        PRODUCT
WHERE       P_PRICE = (SELECT MAX(P_PRICE) FROM PRODUCT);
```

When considering dates, use max for newest date and min for oldest.

SUM

Computes the total sum for a specified attribute, using any conditions imposed. Examples:

```
SELECT      SUM(CUS_BALANCE) AS TOTBALANCE
FROM        CUSTOMER;

SELECT      SUM(P_QOH * P_PRICE) AS TOTVALUE
FROM        PRODUCT;
```

AVG

Outputs the mean average for a specified column or expression, similar in syntax to MIN and MAX

Triggers

Triggers are sections of SQL code associated with a table that perform an action when a row is:

- Inserted
- Updated
- Deleted

Note: triggers may be used because Oracle doesn't support an UPDATE cascade

Syntax/general form:

```
CREATE [OR REPLACE] TRIGGER <trigger_name>
    {BEFORE | AFTER | INSTEAD OF }
    {UPDATE | INSERT | DELETE}
        [OF <attribute_name>] ON <table_name>
    [FOR EACH ROW]
    [WHEN]
DECLARE
BEGIN
        .... trigger body goes here .....
END;
```

Breaking it down:

- The triggering statement specifies the type of SQL statement that fires the trigger body, the possible options and the table associated with the trigger
- For UPDATE, can specify a column list so that it only fires when one of the specific columns is updated (if not included trigger is fired whenever there is an update
- The trigger body (between BEGIN and END) includes SQL statements that are executed when triggering statement is issued
- Two correlation names exist for every table column being modified

Correlation names
For every column value of the current row affected by the triggering statement, we have :OLD.columnmame and :NEW.columnname.

- For DELETE, we only need OLD.columnname
- For INSERT, we only need NEW.columnname
- For UPDATE, both are meaningful

Example:

```
CREATE OR REPLACE TRIGGER Owner_Upd_Cas
BEFORE UPDATE OF owner_no ON owner
FOR EACH ROW
BEGIN
   UPDATE vehicle
   SET    owner_no = :new.owner_no
   WHERE  owner_no = :old.owner_no;
   DBMS_OUTPUT.PUT_LINE ('Corresponding owner number in the
  VEHICLE table has also been updated');
END;
/
```

NB: we want it to affect each changed row to UPDATE CASCADE

10. SQL Part III (Advanced Joins)

Relational Set Operators

UNION

Combines rows from two or more queries w/o including duplicate rows. However, the SELECT statements must return the same number of attributes and similar data types (i.e. UNION compatible)
Syntax:

Example:

```
SELECT      CUS_LNAME, CUS_FNAME, CUS_INITIAL, CUS_AREACODE,
            CUS_PHONE
FROM        CUSTOMER
UNION
SELECT      CUS_LNAME, CUS_FNAME, CUS_INITIAL, CUS_AREACODE,
            CUS_PHONE
FROM        CUSTOMER_2;
```

UNION ALL

Can be used to produce a relation that retains the duplicate rows.

```
SELECT      CUS_LNAME, CUS_FNAME, CUS_INITIAL, CUS_AREACODE,
            CUS_PHONE
FROM        CUSTOMER
UNION ALL
SELECT      CUS_LNAME, CUS_FNAME, CUS_INITIAL, CUS_AREACODE,
            CUS_PHONE
FROM        CUSTOMER_2;
```

INTERSECT

Can be used to combine two queries, returning only the rows from two queries, returning only the rows that appear in both sets.
Syntax:

Example:

```
SELECT      CUS_CODE FROM CUSTOMER WHERE CUS_AREACODE = '615'
INTERSECT
SELECT      DISTINCT CUS_CODE FROM INVOICE;
```

MINUS

Combines rows from two queries and returns only the rows that appear in the first set but not in the second.
Example (we want to know which customers in the CUSTOMER table are NOT found in the CUSTOMER_2 table):

```
SELECT      CUS_LNAME, CUS_FNAME, CUS_INITIAL, CUS_AREACODE,
            CUS_PHONE
FROM        CUSTOMER
MINUS
SELECT      CUS_LNAME, CUS_FNAME, CUS_INITIAL, CUS_AREACODE,
            CUS_PHONE
FROM        CUSTOMER_2;
```

Views

A view is a virtual table based on a SELECT query that is saved as an object in the DB. It can contain columns, computed columns, aliases and aggregate functions.
Note: the base table is the table on which a view is based.

Syntax:

```
CREATE VIEW viewname AS SELECT query
```

Characteristics of Views

- We can use a view in place of a table in an SQL statement
- Views are dynamically updated (recreated each time it is called)
- Views provide a level of security as it restricts what can be seen
- Views can be the basis of reports

JOINS

Note: use the following pair of tables for the examples here:

Student

ID	NAME
1	Alice
2	Bob
3	Chris

Mark

ID	SUBJECT	MARK
1	1004	95
2	1045	55
1	1045	90
4	1004	100

Natural Join

Returns only the rows with matching values in the matching columns (matching columns must have the same name and similar data type. Example:

Natural Join gives no information for Chris (no enrolment, e.g. just enrolled)
and the student with ID 4 (student without info, e.g. quit uni)

ID	NAME	ID_1	SUBJECT	MARK
1	Alice	1	1004	95
2	Bob	2	1045	55
1	Alice	1	1045	90

```
select * from student s JOIN mark m on s.id = m.id;
```

FULL OUTER JOIN

Returns rows with matching values and includes rows from both tables with unmatched values

Get (incomplete) information of both Chris and student with ID 4
i.e. we want NULLS both sides too.

ID	NAME	ID_1	SUBJECT	MARK
1	Alice	1	1004	95
2	Bob	2	1045	55
1	Alice	1	1045	90
(null)	(null)	4	1004	100
3	Chris	(null)	(null)	(null)

```
select * from
student s FULL OUTER JOIN mark m
on s.id = m.id;
```

LEFT OUTER JOIN

Returns rows with matching values and includes all rows from left table with unmatched values

Get (incomplete) information of only Chris - i.e. make sure all LEFT values present...

ID	NAME	ID_1	SUBJECT	MARK
1	Alice	1	1004	95
2	Bob	2	1045	55
1	Alice	1	1045	90
3	Chris	(null)	(null)	(null)

```
select * from
student s LEFT OUTER JOIN mark m
on s.id = m.id;
```

RIGHT OUTER JOIN

Returns rows with matching values and includes all rows from the right table with unmatched values

ID	NAME	ID_1	SUBJECT	MARK
1	Alice	1	1045	90
1	Alice	1	1004	95
2	Bob	2	1045	55
(null)	(null)	4	1004	100

```
select * from
student s RIGHT OUTER JOIN mark m
on s.id = m.id;
```

11. DB Interface

Understanding of the principles and ALL core concepts:

- Database middleware
- Web to database middleware
    - Database middleware: manages connectivity and data transformation issues (eg. MS ODBC, JDBC)
    - Web-to-database middleware: Program that interact directly with the server process to handle specific types of requests

        - Web server <> Web-to-database-middleware <> Database middleware <> Database
- Using PHP to communicate with databases
    - PHP is a server- side language that can connect to the Oracle database using the OCI8 extension
    - oci_connect used to connect to an Oracle Database

Database design frameworks

- Modern frameworks
    - Web frameworks: Provides easier development of web apps
        - Also support Oracle connectivity
- Object-Relational Mapping: Allows queries and data manipulation with a database using OOP
- Security
    - SQL Injection: Inserting an SQL query via input from the client to the app for harmful use
    - Prevented by sanitising and checking the input

Extra Stuff

SQL Data Definition Commands

COMMAND OR OPTION	DESCRIPTION
CREATE SCHEMA AUTHORIZATION	Creates a database schema
CREATE TABLE	Creates a new table in the user's database schema
NOT NULL	Ensures that a column will not have null values
UNIQUE	Ensures that a column will not have duplicate values
PRIMARY KEY	Defines a primary key for a table
FOREIGN KEY	Defines a foreign key for a table
DEFAULT	Defines a default value for a column (when no value is given)
CHECK	Validates data in an attribute
CREATE INDEX	Creates an index for a table
CREATE VIEW	Creates a dynamic subset of rows and columns from one or more tables (see Chapter 8, Advanced SQL)
ALTER TABLE	Modifies a table's definition (adds, modifies, or deletes attributes or constraints)
CREATE TABLE AS	Creates a new table based on a query in the user's database schema
DROP TABLE	Permanently deletes a table (and its data)
DROP INDEX	Permanently deletes an index
DROP VIEW	Permanently deletes a view

SQL Data Data Manipulation Commands

COMMAND OR OPTION	DESCRIPTION
INSERT	Inserts row(s) into a table
SELECT	Selects attributes from rows in one or more tables or views
WHERE	Restricts the selection of rows based on a conditional expression
GROUP BY	Groups the selected rows based on one or more attributes
HAVING	Restricts the selection of grouped rows based on a condition
ORDER BY	Orders the selected rows based on one or more attributes
UPDATE	Modifies an attribute's values in one or more table's rows
DELETE	Deletes one or more rows from a table
COMMIT	Permanently saves data changes
ROLLBACK	Restores data to its original values
Comparison operators	
=, <, >, <=, >=, <>, !=	Used in conditional expressions
Logical operators	
AND/OR/NOT	Used in conditional expressions
Special operators	Used in conditional expressions
BETWEEN	Checks whether an attribute value is within a range
IS NULL	Checks whether an attribute value is null
LIKE	Checks whether an attribute value matches a given string pattern
IN	Checks whether an attribute value matches any value within a value list
EXISTS	Checks whether a subquery returns any rows
DISTINCT	Limits values to unique values
Aggregate functions	Used with SELECT to return mathematical summaries on columns
COUNT	Returns the number of rows with non-null values for a given column
MIN	Returns the minimum attribute value found in a given column
MAX	Returns the maximum attribute value found in a given column
SUM	Returns the sum of all values for a given column
AVG	Returns the average of all values for a given column

Comparison Operators

SYMBOL	MEANING
=	Equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
<> or !=	Not equal to

Arithmetic Operators

OPERATOR	DESCRIPTION
+	Add
-	Subtract
*	Multiply
/	Divide
^	Raise to the power of (some applications use ** instead of ^)