SQL: Chapter 1: Introduction

The objective of this chapter is to introduce the main concepts of data storage and retrieval in the context of database information systems.

In view of their prominence this booklet concentrates on the general characteristics of Relational Database Management Systems (RDBMS) and the Structured Query Language SQL and does not consider any of the numerous other types of databases. No prior knowledge of SQL is assumed.

It is intended that the SQL presented in this booklet be followed interactively and that the you should try all the given examples in the order in which they are presented. At the end of the booklet you should have attained a thorough knowledge of SQL and its capabilities as an interactive statement language.

In the main the SQL covered complies with the standard definition for SQL were proprietary SQL features are referred to this will be made clear. On this basis the skills obtained from this unit should be transferable across a wide range of RDBMS's which support SQL.

To enable you to make effective use of your local facilities a number of RDBMS specific appendices containing access instructions and other supplementary information specific to your environment have been included.

SQL: Chapter 2: Files, Databases and Database Management Systems

----

2. Introduction

Data and its storage may be considered to be the heart of any information system. Data has to be up-to-date, accurate, accessible in the required form and available to one or perhaps many users at the same time.

For data to be of value it must be presented in a form which supports the various operational, financial, managerial, decision making, administrative and clerical activities within an organisation.

To meet these objectives data needs to be stored efficiently - to avoid lengthy access times - and with minimal duplication - to avoid lengthy update times and the possibility of inconsistency and inaccuracy. For the data stored by a given organisation to have any value at all its integrity (consistency and accuracy) must always be assured.

In this section we are going to consider what is know as the conceptual view (user view) of stored data. As such we do not need to be concerned with how data is physically stored on specific types of storage media; neither do we need to consider the various storage structures and access methods applicable to retrieval of data from such media.

2.1 Data Storage.

The data of an organisation takes the form of an abstract representation of objects and events which occur within the organisation's environment. Within the context of an airline company for example, such objects might include Aircraft, Passengers and Airports, and include events such as Flights and the issuing of Tickets.

There are two principle approaches to the storage of data in a computer-based information system. Data may be stored in separate files belonging to isolated information systems operating within individual departments, or data may be stored in a database which may serve as a resource available across all departments and functional areas. The following sections consider these two approaches and their relative merits.

2.2 File Based Systems.

Before we consider the issues arising from file based storage let us firstly, establish some general concepts and definitions relating to the storage of data in files.

2.2.1 Data Files.

A file is a complete, named collection of information and the basic unit of storage that enables a computer to distinguish one set of information from another. For example a file named "Aircraft" might contain information about the different types of aircraft used by a particular airline company and a file named "Airport" might hold details of all the airports from which the airline operates.

2.2.2 Records.

The data held within a file are organised into structured groups of related elements called records. For example, a record describing an individual aircraft might be composed of the data elements: "identifying number", "name of manufacturer"; "description", "classification" (turbo-prop, jet, etc) "seating capacity" and so on. The aircraft file then contains zero, one, or many such records; where each record describes an individual aircraft.

2.2.3 Fields.

The individual elements of a record are referred to as fields. Hence form the example above, "identifying number", "name of manufacturer", "description", "classification" and "seating capacity", each represent an individual field (element) of the aircraft record.

2.2.4 Data Types.

The data to be held within each field of a given record will possess certain characteristics in terms of size (length measured in characters or digits) and type (numeric, alphabetic, dates, etc). Each field of a record is allocated a particular data type which describes the allowed characteristics of the data to be held by the field and further indicates the range of operations which can be carried out on the field; for example, arithmetic operations would be valid on fields containing numeric data but not on fields containing an address or a narrative description.

2.2.5 Keys.

A key is a field or combination of fields used to identify a record. When a key uniquely identifies a record it is referred to as the primary key.

Continuing with the example of the Aircraft record, if a given value of the field "identifying number" identifies an individual aircraft then it could serve as the primary key. Other fields such as "manufacturer" for example, could also serve as alternative keys (secondary keys) by which a set of records (eg all aircraft from a particular manufacturer) could be identified.

2.3 Issues Arising From File Based Systems.

2.3.1 Data Duplication.

The use of individual data files each serving separate information system tends to lead to situations in which an organisation maintains many copies of the same basic information.

For example let us consider a sales department which calculates bonuses payable to individual sales personnel on the value of their sales; with actual bonus payments being made only after confirmation that the sales invoices on which bonuses have been calculated have been fully paid.

If the sales department was to operate its bonus scheme based on its own sales and receipts files then the sales department would undoubtedly be holding copies of data such as employee name, payroll number, invoice numbers and amounts received, which would already be recorded elsewhere in the organisation by the personnel department and by the accounts department for example.

A change in marital status resulting in a change of surname would therefore result in the need to update several files and queries against invoices resulting in debit or credit notes being raised would also require the updating of several files.

2.3.2 Data Inconsistency and Integrity.

Where several discrete files exist serving the immediate requirements of individual departments there is a strong likelihood that the common or duplicated data held by these files will get out of step with each other resulting in different versions of data being held by the organisation. If the accounts receivable system in the example above issues a credit note against an incorrectly totalled invoice and fails to advise the sales department of this then the integrity of the sales department's files would be lost and as a result incorrect bonus payments would be made.

2.3.3 File Design.

Not withstanding the above consequences of file based information systems, taken from the view point of an individual information system, it is a relatively easy matter to ensure that the required files are designed to "perfectly" suite user/application needs. As such, taken in isolation, such information systems are capable of presenting information in exactly the form required by their users and also of providing highly efficient usage of storage and rapid retrieval times.

2.4 Databases and Database Management Systems.

For a generalised consideration of databases we may continue to apply the data storage concepts (files, records, keys, etc) previously introduced for file based systems. However, databases are not just a collection of files; through specific access controls provided by the Database Management System (DBMS) databases are able to provide a central resource of data which can be shared between users on an organisation wide basis.

2.4.1 Database Design.

To satisfy the information needs of users across an organisation the database has to be designed (in terms of "files" and "record layouts") in a generalised manner.

Considering the personnel and sales departments views of personnel records for example. The sales department is only interested in a specific type of personnel record; those records for sales men and women. The sales department might also only be interested in fields such as payroll number, name and bonus. Whilst the personnel department on the other hand would be interested in all personnel records irrespective of job description (a field in the personnel record) and would want record fields in addition to those of interest to the sales department, such as home-address, marital status, date-of-appointment, department, salary-scale and scale-point, national insurance number and so on.

2.4.2 User Views.

It is the role of the DBMS to provide facilities which enable data (from a generalised definition) to be presented in the form required by specific users. So for example the DBMS should provide the sales department with just those record fields which they require about sales personnel whilst at the same time providing the personnel department with their requirements.

2.4.3 Database Security and Integrity.

There are two major consequences arising from the shared usage of data, namely, security and integrity.

Firstly, the DBMS must ensure that users are only allowed access to data which they are authorised to access. In addition, access authorisation must also restrict the type of access; limiting some users to read-only access for certain instances of data for example. So, if only the accounts receivable users are permitted to adjust the value of invoices, by the issuing of credit notes for instance, then the sales department, whilst interested to see when full payment of an invoices has been received, should not be permitted to change the invoice records in any way.

Secondly, the DBMS must also ensure that conflicting updates do not occur. In a stock control systems for example, a user updating the database for a customer order, must be given exclusive access to the records of the ordered items so their "quantity on hand" fields may be updated. Such exclusive access should prevent anyone from looking at out of date quantity on hand figures and should also prevent two or more users from trying to update the same quantity on hand values at the same time. This serialisation of record updating is controlled through the DBMS, which in the case of the above example, would issue locks on the records required by each customer order transaction.

2.4.4 Database Performance.

Because databases maintain data in a generalised form, converting this generalised form into a series of user views as required, they are generally less efficient in terms of data storage usage and in terms of access times compared with their individual counterpart file based systems.

SQL: Chapter 3: Relational Databases

The Relational Model   Relational Database Terminology   Database Tables and Relationships

----

3.1 The Relational Model.

The relational model was developed by E F Codd at the IBM San Jose Research Laboratory in the late 1960s. This work being published in 1970 under the title:

"A Relational Model of Data For Large Shared Data Banks".

In this paper Codd defines the relational model and its capabilities mathematically.

Following this publication a number of research projects were undertaken in the early 1970s with the aim of implementing a relational database management system. The earliest of these projects included, System R at IBM, San Jose and Ingres at the University of California, Berkeley.

3.1.1 Relational Query Languages.

The relational database model as defined by Codd included a number of alternative relational query languages.

The Ingres project developed a query language called Quel which broadly complies with Codd's definition of a tuple relational calculus query language. Quel is still a part of the Ingres DBMS available today; although in view of current trends SQL is generally chosen.

The System R project developed a series of query languages; the first of these called SQUARE, was later developed into a more convenient form called SEQUEL. SEQUEL was itself further developed into the form of today's SQL.

In 1986 the American National Standards Institute ANSI published an SQL standard the:

"Systems Application Architecture Database Interface (SAA SQL)".

Query languages have two main components:

Data Manipulation Language (DML), and

Data Definition Language (DDL);

where the DML part of the language is used to retrieve, delete and amend instances of data in the database and where the DDL part of the language is used to describe the type of data to be held by the database.

3.2 Relational Database Terminology.

In section 2 you were introduced to some general concepts and definitions relating to the storage and retrieval of data. In a Relational Database all data may be viewed in the form of simple two-dimensional tables and to distinguish this representation of data from that of other representations we use a separate terminology to describe the data held in a Relational Database.

There are in fact alternative terms used to describe the data in a relational database. The first is taken from the formal definition of the relational model and the second is based on the ability to view data in the form of simple tables. We will adopt the latter terminology for the remainder of this booklet.

Equivalent Terms

Relational Databases		Non Database
Relation	Table	File
Tuple	Row	Record
Attribute	Column	Field

Where "tuple" is pronounced tup^el (as in couple).

3.3 Database Tables and Relationships.

The table definitions for an "airline" database are given in detail in Appendix A1 and summarised here.

Table Name	Role
Airport	Records all airports from which the Airline operates.
Route	Records all routes flown by the Airline.
Fares	Contains details of different classes of Fare which are available, eg Standby Single, Business Return, APEX, etc.
Tariff	Details the prices of fares as applicable to each of the operated routes.
Aircraft	Records each type of aircraft used by the Airline.
Flight	Holds details of all timetabled flights, times of departure, destination, and service.
Passenger	Records all passengers who have flown with the airline.
Ticket	Records tickets currently allocated to passengers.

Itinerary   Details for a given ticket, all flights and
flight-dates in order.

The data held by these tables do not exist independently. Hence, there are a number of inter-relationships which must be considered. The relationships between the tables in the "airline" database are as follows:

A Passenger may have more than one Ticket.

A Ticket comprises an Itinerary of one or more Flights.

A Flight occurs between two Airports (departs from - arrives at).

Each Flight has a specific type of Aircraft allocated to it; a given type of Aircraft may be allocate to a number of different flights.

Each Flight serves a particular Route; a Route being served by several Flights occurring at different times of the day.

Routes may be domestic or continental, may appeal to different sorts of passengers (business, excursions), etc and therefore each Route has a different range of Fares and Tariffs.

In a relational database, relationships such as these are implemented via so called foreign keys.

Any column of a given table constitutes a foreign key if it can contain value which refers to a single rows of another table; ie if the given column contains the value of the primary key of another table. In the following example Itinerary. TicketNo is a foreign key which supports the one-to-many relationship between "ticket" and "itinerary"; ie a ticket comprises an itinerary of one or more flight legs.

TICKET

TICKETNO	TICKETDATE	PID
100001	01-07-95	26

100002	25-08-95	28
100010	09-08-95	29
100011	11-08-95	24
100012	01-09-95	21

ITINERARY(example data of flight legs on ticket 100012)

TICKETNO	FLIGHTNO	LEGNO	FLIGHTDATE	FARETYPE
100012	BD776	1	15-09-92	SDR
100012	BD655	2	15-09-92	APR
100012	BD652	3	20-09-92	APR
100012	BD775	4	20-09-92	SDR

In Appendix A1 the primary key column(s) of each table appear underlined. (If you look closely you will notice that these columns also appear in other tables not underlined; where they serve as a foreign key enabling one table to be referenced from another.)

The ability to reference one table from another enables us to implement relationships between one table and another.

For example if we wanted to find the seating capacity of the aircraft allocated to a particular flight we would take the value of AircraftType for that flight, eg DC9, and look for the single row in the Aircraft table with a key value of DC9 to find the seating capacity for that aircraft. Conversely if we wanted to find all flights which have been allocated with a DC9 aircraft we would list from the Flight table all rows with a value of AircraftType equal to DC9.

The following exercise is designed to help you familiarise yourself with the tables of the "airline" database and the relationships between them.

Desk Exercise.

Turn to the database schema and table contents given in Appendix A1 and attempt the following questions:

1) Each airport is identified by a short code. What are the identification codes for Heathrow, Leeds/Bradford and Brussels?

2) The airline provides flights on a number of different Routes; what is the description for RouteNo 9.

3) On any given route the airline offers a number of different types of fare, "Standby Single", "Eurobudget", etc. Find and list the complete range of "Return" Fares offered by the Airline.

4) What is the Tariff (Price) on Route 9 of an "Advanced Purchase Return"?

5) What is the seating capacity (NoSeats) of the aircraft allocated to FlightNo BD412?

6) What are the names (AName) of the airports (FromAirport-ToAirport) used on FlightNo BD80?

7) List the Itinerary (FlightNo, LegNo, FlightDate) for Ticket No 100001. What is the Fare for this ticket and how much would the ticket cost?

8) List the names (AName) of the airports which will be visited by passenger

R H Miller.

<Image>

SQL: Chapter 4: SQL Data Manipulation Language

Introduction Selecting Columns Selecting Rows Ordering the Rows of the Result Grouping the Rows of the Result

----

4. Introduction

SQL is a non-procedural language that is, it allows the user to concentrate on specifying what data is required rather than concentrating on the how to get it.

The non-procedural nature of SQL is one of the principle characteristics of all 4GLs - Fourth Generation Languages - and contrasts with 3GLs (eg, C, Pascal, Modula-2, COBOL, etc) in which the user has to give particular attention to how data is to be accessed in terms of storage method, primary/secondary indices, end-of-file conditions, error conditions (eg ,Record NOT Found), and so on.

The DML component of SQL comprises four basic statements:

SELECT to retrieve rows from tables

UPDATE to modify the rows of tables

DELETE to remove rows from tables

INSERT to add new rows to tables.

Sections 4.1 through 4.5 illustrate with examples the data retrieval capabilities of the SELECT statement as well as introducing more general features of SQL such as arithmetic and logical expressions which are equally as applicable to the UPDATE, DELETE and INSERT statements which are dealt with separately in section 4.6.

Each section contains examples which illustrate the various options available with each statement. You are advised to try out these examples for yourself. In addition, at the end of each section there are a number of exercise queries designed to test your understanding of the section.

Before proceeding further you should now refer to the appendices which explain how to access to your local database facilities.

If you have any difficulties at this point ask your tutor for help.

4.1 Selecting Columns.

The simplest form of the SELECT statement consists of just two clauses.

```
SELECT column-list

FROM table-list ;
```

4.1.1 Specifying Required Column List.

Example 4.1.1 - List full details of all Aircraft.

In order to list all the details (columns) recorded for each aircraft type we will list the name of each column in the Aircraft table as follows.

```
SELECT AircraftType, ADescription, NoSeats

FROM Aircraft ;
```

Result:

```
----------------------------------------------------------------------



aircra  adescription           noseat
```

```
DC9     Advanced Turbo Prop    48
ATP     McDonnel Douglas Jet   120
737     Boeing 737-300 Jet     300
```

```
----------------------------------------------------------------------
```

For those queries in which all columns of a particular table are to be retrieved the column-list may be replaced by an asterisk. Hence:

```
SELECT *

FROM Aircraft ;
```

gives the same result as before.

Example 4.1.2 - List the Names and Addresses of all Passengers.

This query requires the selection of individual columns from the Passenger table and so we simply list the columns required as follows:

```
SELECT Name, Address

FROM Passenger ;
```

Result:

```
----------------------------------------------------------------------



name           address

J Millar       Englewood Cliffs
J D Ullman      1 Microsoft Way
A Smithson      16 Bedford St
D Etheridge    4 Maylands Avenue
E Simon        8 Cherry Street
D N Hamer       1 St Paul's
D E Avison     Churchyard
G B Davis      5 Chancery Lane
C Evans        25 Allenby Road
A N Smith      63 Kew Green
T Pittman      81 Digby Crescent
J Peters       The Little House
K E Kendall    31 Lucas Road
R H Miller     11 Rosedale Avenue
             155 Kingston Road
```

4.1.2 Removing Duplicate Rows.

Example 4.1.3 - List all Flight No's with tickets issued against them.

Flights for which tickets have been issued are recorded in the Itinerary table. The following simple query lists Flight No's from the Itinerary table.

```
SELECT FlightNo

FROM Itinerary ;
```

Result:

```
----------------------------------------------------------------------



flight

BD80
BD95
BD80
BD95
BD82
BD54
BD776
BD655
BD652
BD775
BD772
BD655
BD652
BD412
BD419
BD412
BD419
BD224
BD255
BD256
BD275
BD412
BD582
BD589
BD332
BD51
BD774
```

```
BD659
BD658
BD771
BD54




----------------------------------------------------------------------
```

31 rows retrieved

However, a given FlightNo will appear as many times as the number of tickets issued for that flight, therefore the result of this query contains duplicate rows. To remove duplicates from the result of this sort of query SQL provides the DISTINCT function which is used as follows.

```
SELECT DISTINCT FlightNo

FROM Itinerary ;
```

Result:

```
----------------------------------------------------------------------



flight

BD224
BD225
BD256
BD275
BD332
BD412
BD419
BD51
BD54
BD582
BD589
BD652
BD655
BD658
BD659
BD771
BD772
BD774
BD775
BD776
BD80
BD82
```

```
BD95

-----------------------------------------------------------------------

23 rows retrieved
```

4.1.3 Arithmetic Expressions.

SQL allows arithmetic expressions to be included in the SELECT clause. An arithmetic expression consists of a number of column names and values connected by any of the following operators:

+ Add

- Subtract

* Multiply

/ Divide

When included in the SELECT clause the results of an expression are displayed as a calculated table column.

Example 4.1.4 - Assuming Tariffs are recorded excluding VAT, calculate and list Tariffs inclusive of VAT.

The are a number of different ways of calculating Fare prices inclusive of VAT (17.5%). The following are all equivalent and valid arithmetic expressions for calculating VAT inclusive values within the SELECT clause.

i) Price + Price * .175

ii) Price + Price * 17.5 / 100

iii) Price * 1.175

```
SELECT RouteNo, FareType, Price, Price*1.175

FROM Tariff ;
```

Result:

See next page.

```
-----------------------------------------------------------------------


routen   farety    price      col4
```

```
   3 BUR     $117.00   $137.48
   3 SDR     $158.00   $185.65
   3 SDS      $79.00    $92.83
   4 SDR     $162.00   $190.35
   4 SBS      $49.00    $57.58
   6 BUR     $117.00   $137.48
   6 SBS      $42.00    $49.35
   6 KFS      $53.00    $62.28
   7 SDR     $128.00   $150.40
   8 SDS      $74.00    $86.95
   9 PXR     $153.00   $179.78
   9 EUR     $181.00   $212.68
   9 APR      $95.00   $111.63
  11 KFS      $59.00    $69.33
  13 EXR     $121.00   $142.18
  14 SDR     $110.00   $129.25
  14 SBS      $33.00    $38.78
  15 SBS      $33.00    $38.78
```

To obtain a column name for the result column enter the following statement:

```
SELECT RouteNo, FareType, Price, Price*1.175 as vat_price

FROM Tariff ;
```

(The result-column name must not have embedded spaces and it does not have quotes around it.)

Ingres applies standard operator precedence ie:

Multiply (*) and divide (/) are evaluated before plus (+) and minus (-).

Multiply and divide are equal in precedence and plus and minus are equal in precedence.

Operators of equal precedence are evaluated in order of appearance from left to right. Hence:

6 + 4 / 2 ... evaluates to 8 (6+2) and not 5 (10/2).

The order of evaluation may be controlled by the use of brackets. If in the above example, we had wanted the + operator to be evaluated first then the following use of brackets would make the expression

(6 + 4) / 2 ... evaluates to 5.

4.1.4 Aggregate (Group) Functions.

SQL provides a number of special functions called aggregate functions which may be included in the SELECT clause. Each of these functions operates over a named column and returns (calculated from all

selected rows) a single value.

AVG(column-name) returns the average value

COUNT(column-name) returns the number of

non-null values

COUNT(DISTINCT column-name) returns the number of distinct values

COUNT(*) returns the number of rows

MAX(column-name) returns highest value

MIN(column-name) returns the lowest value

SUM(column-name) calculates the total of values

Example 4.1.5 - List the average aircraft seating capacity.

SELECT AVG(NoSeats)

FROM Aircraft ;

Result:

```
----------------------------------------------------------------------


col1

156.00




----------------------------------------------------------------------
```

Example 4.1.6 - Count the number of Routes flown by the airline.

SELECT COUNT(*)

FROM Route ;

Result:

```
----------------------------------------------------------------------
```

```
col1

10
```

Aggregate functions return a single valued result, ie a calculated column with only one row. As such these functions cannot be included in the SELECT clause with columns which are multi-valued, ie resulting in a two or more rows. On this basis the following query is illegal and would not be processed by the DBMS.

```
SELECT AircraftType, AVG(NoSeats)

FROM Aircraft ;
```

Later on when we have extended our consideration of SELECT to accommodate slightly more complicated queries we will see examples of how to use aggregate functions with "group columns" in the SELECT clause.

Exercise 1.

Give the SQL required to

1. List the full details of all Airports.

2. List the distinct AircraftTypes assigned to flights.

3. List all Tickets with TicketDate appearing in the left most column.

4. From amongst all of the different aircraft types find the largest seating capacity. (You are not asked to find which particular aircraft type this seating capacity belongs to.)

5. Calculate how many different AircraftTypes are allocated to timetabled flights by querying the flights table.

6. Format the query given in Example 4.1.4 so that the expression price*1.175 is given a result-column name of vat_price

4.2 Selecting Rows.

In the previous section we were concerned only with listing one or more columns from a table. In this type of query all rows of the queried table are returned. More usually however, we will only be interested in one or a small number of rows from a given table, those which satisfy a particular condition, for example if we wanted to find the number of seats on a McDonnell Douglas DC9 Jet then only the one row in the Aircraft table which records this type of aircraft would be of interest.

In order to select specific rows from a table we use the WHERE clause which is placed after the FROM as follows.

SELECT column-list

FROM table-list

WHERE conditional-expression ;

4.2.1 Specifying Selection Conditions.

When you use a WHERE clause, you are specifying a condition for SELECT to test when processing each row of the table. Only those rows which test True against the condition are listed in the result.

Example 4.2.1 - Find the Seating Capacity on a DC9.

SELECT NoSeats

FROM Aircraft

WHERE AircaftType = 'DC9' ;

Result:
----

noseat

120

This illustrates the most commonly used form of the WHERE clause in which the expressions consists of three elements:

1.a column name AircraftType 2.a comparison operator = {equal to} 3.a column name, constant value, or list of values 'DC9'

Comparison Operators.

Any of the following comparison operators may be used:

= equal to

<> (!=) not equal to

> greater than

< less than

>= greater than or equal to

<= less than or equal to

A number of special comparison operators are also provide:

BETWEEN Compares a range of values.

IN Tests against values in a list.

ANY Used in conjunction with comparison

ALL operators to test against values in a list.

LIKE Compares a character pattern.

Constants.

Where a constant contains one or more alphabetic characters it is distinguished from a column-name by placing the constant between apostrophes.

List of Values.

See section 4.2.3 Matching a Value in a List.

Example 4.2.2 - List Descriptions of Aircraft with more than 50 seats.

```
SELECT ADescription, NoSeats

FROM Aircraft

WHERE NoSeats > 50 ;
```

Result:

```
----------------------------------------------------------------------



adescription            noseat

McDonnel Douglas Jet    120
Boeing 737-300 Jet      300
```

Expressions formed from these comparison operators are known as logical expressions because they evaluate to one of two possible logic states True or False.

In the above example, if a given row has a value in the Noseats column which is greater than 50 then the expression will evaluate to true, if the value is less than or equal to 50 the expression will then evaluate to false. Only rows which cause the WHERE clause expression to evaluate to true are listed in the result.

4.2.2 Selecting Rows From a Range.

The BETWEEN operator provides a convenient way of selecting rows with a column value which is within a specified range.

Example 4.2.3 - List the names and addresses of passengers with Pid's in the range 25 to 35 inclusive.

```
SELECT Pid, Name, Address

FROM Passenger

WHERE Pid BETWEEN 25 AND 35 ;
```

Result:

```
----------------------------------------------------------------------



pid    name          address

26     J Millar      Englewood Cliffs
28     J D Ullman     1 Microsoft Way
29     A Smithson     16 Bedford St
30     D Etheridge    4 Maylands Avenue
34     E Simon        8 Cherry Street





----------------------------------------------------------------------
```

If required the NOT operator may be used to negate the result of the BETWEEN comparison. Use of the NOT operator in this case would specify passengers whose Pid's are less than 25 or greater than 35, ie

```
SELECT Pid, Name, Address

FROM Passenger

WHERE Pid NOT BETWEEN 25 AND 35 ;
```

Result:

```
-----------------------------------------------------------------------



pid   name             address

10    D N Hamer        1 St Paul's
20    D E Avison      Churchyard
21    G B Davis       5 Chancery Lane
24    C Evans         25 Allenby Road
90    A N Smith       63 Kew Green
91    T Pittman       81 Digby Crescent
92    J Peters       The Little House
93    K E Kendall     31 Lucas Road
94    R H Miller      11 Rosedale Avenue
                   155 Kingston Road





-----------------------------------------------------------------------
```

4.2.3 Matching a Value in a List.

The IN operator permits the selection of rows with a column value which matches any value from a set of values.

Example 4.2.4 - Find the Tickets (No's) issued on Flight Nos BD54, BD80, BD412, BD582, or BD332.

```
SELECT TicketNo, FlightNo

FROM Itinerary

WHERE FlightNo IN ('BD54','BD80','BD412', 'BD582', 'BD332') ;
```

Result:

```
-----------------------------------------------------------------------



ticketno  flight

100001    BD80
100002    BD80
100011    BD54
100021    BD412
100022    BD412
100041    BD412
```

```
100051    BD582
100052    BD332
100100    BD54
```

```
----------------------------------------------------------------------
```

(See the equivalent boolean expression in Example 4.2.8)

The NOT operator may be used to negate this result, ie

```
SELECT TicketNo, FlightNo

FROM Itinerary

WHERE FlightNo NOT IN ('BD54','BD80','BD412', 'BD582','BD332' );
```

Result:

```
----------------------------------------------------------------------


ticketno  flight

100001    BD95
100002    BD95
100010    BD82
100012    BD776
100012    BD655
100012    BD652
100012    BD775
100020    BD772
100020    BD655
100020    BD652
100021    BD419
100022    BD419
100030    BD224
100030    BD255
100030    BD256
100030    BD275
100051    BD589
100100    BD51
100100    BD774
100100    BD659
100100    BD658
100100    BD771
```

Note equivalencies between the IN and ANY operators:

= ANY (...) = IN (...)

!= ANY (...) = NOT IN (...)

4.2.4 Matching a Character Pattern.

The comparison operators = and IN match exact values, however there are occasions when an exact value for comparison is not known or where or a partial match only is all that is required.

In these situations the LIKE operator allows rows to be selected that partially match a given pattern of characters.

The LIKE operator recognises two special character symbols

% which represents any sequence of zero or more characters

_ (underscore) which represents any single character.

Example 4.2.5 List the description of all Routes using Birmingham.

For this query we are interested in any Route which includes the character constant 'Birmingham' anywhere within its description.

```
SELECT *

FROM Route

WHERE RDescription LIKE '%Birmingham%' ;
```

Result:

```
----------------------------------------------------------------------


routeno   rdescription

9         Birmingham-Brussels
14         Heathrow-Birmingham
```

Example 4.2.6 - Find the telephone numbers of all passengers with a surname of either 'Miller' or 'Millar'.

```
SELECT Name, TelNo

FROM Passenger

WHERE Name LIKE '%Mill_r' ;
```

Result:

```
-----------------------------------------------------------------------



name              telno

J Millar        061 343 881
R H Miller         0638 4672





-----------------------------------------------------------------------
```

4.2.5 Compound Expressions with Boolean Operators.

So far we have only considered the WHERE clause composed of single logical expressions. There are many occasions however when selections need to be based on several conditions. There may be a number of alternative conditions by which rows can be selected or there may be a number of conditions which must be satisfied by all selected rows.

Individual logical expressions may be combined within the WHERE clause through the use of the two Boolean operators:

AND

OR

Expressions are then combined as follows:

```
WHERE logical-exp AND logical-exp AND ...

logical-exp OR logical-exp ...
```

Example 4.2.7 - List the names and addresses of passengers with Pid's in the range 25 to 35 inclusive.

```
SELECT Pid, Name, Address
```

```
FROM Passenger

WHERE Pid >= 25 AND Pid <= 35 ;
```

```
Result:
----------------------------------------------------------------------



pid   name           address

26    J Millar       Englewood Cliffs
28    J D Ullman     1 Microsoft Way
29    A Smithson     16 Bedford St
30    D Etheridge    4 Maylands Avenue
34    E Simon        8 Cherry Street





----------------------------------------------------------------------
```

Logical Operator Precedence.

Comparison operators (=, <, IN, LIKE, etc) are evaluated first, then the AND operators then the OR operators. As with arithmetic operators brackets may be used to change the order of evaluation. Contents of brackets being evaluated first.

If we take the query from Example 4.2.7 and consider three rows from the Passenger table with values in the Pid column of 36, 24 and 30 respectively, then given the above precedence rules, they would be evaluated as summarised in the table below.

Pid	Pid >=25	Pid <= 35	AND	-- Result --
36	true	false	false	not selected
24	false	true	false	not selected
30	true	true	true	selected

As can be observed, for the AND operator to evaluate to true both of the tested conditions must also be true.

The action of the AND and OR operators are summarised in the following tables in which 1 represents true and 0 represents false.

AND-Operator	OR-Operator
0 AND 0 = 0	0 OR 0 = 0

1 AND 0 = 0 1 OR 0 = 1

0 AND 1 = 0 0 OR 1 = 1

1 AND 1 = 1 1 OR 1 = 1

Example 4.2.8 - Find the Tickets (No's) issued on Flights BD54, BD80, BD412, BD582, or BD332.

In Example 4.2.4 we used the IN operator for this query; this is the equivalent SQL using the OR operator.

```
SELECT TicketNo, FlightNo

FROM Itinerary

WHERE FlightNo = 'BD54'

OR FlightNo = 'BD80'

OR FlightNo = 'BD412'

OR FlightNo = 'BD582'

OR FlightNo = 'BD332' ;
```

Result:

```
----------------------------------------------------------------------


ticketno  flight

100001    BD80
100002    BD80
100011    BD54
100021    BD412
100022    BD412
100041    BD412
100051    BD582
100052    BD332
100100    BD54


----------------------------------------------------------------------
```

The WHERE clause may be contain a combination of different types of expressions as illustrated in the following example.

Example 4.2.9 - List the names and addresses of passengers who either have Pid's in the range 25 to 35 inclusive or have a surname of 'Smith'.

```
SELECT Pid, Name, Address

FROM Passenger

WHERE Pid >= 25 AND Pid <= 35

OR Name LIKE '%Smith' ;
```

Result

```
----------------------------------------------------------------------


pid   name          address

26    J Millar      Englewood Cliffs
28    J D Ullman    1 Microsoft Way
29    A Smithson    16 Bedford St
30    D Etheridge   4 Maylands Avenue
34    E Simon       8 Cherry Street
90    A N Smith     81 Digby Crescent




----------------------------------------------------------------------
```

4.2.6 Arithmetic Expressions.

SQL allows arithmetic expressions to be included as part of the WHERE clause.

Example 4.2.10 - List the Tariffs available on Route No 9 (Birmingham-Brussels) which when VAT (@17.5%) is added cost more than $170.00

```
SELECT FareType, Price, Price*1.175

FROM Tariff

WHERE RouteNo = 9

AND (Price*1.175) > 170 ;
```

Result:

```
----------------------------------------------------------------------
```

```
faretype  price     col3

PXR       $153.00   $179.78
EUR       $181.00   $212.68
```

Exercise 2.

Give the SQL required to:

1. List the FlightNo's on the itinerary for Ticket 100030.

2. Calculate the total ticket price for Ticket 100030 given the Tariff for each flight is $50.00. Assign a suitable title to your result-column.

3. List full details of flights where the allocated aircraft is either a DC9 or a 737.

4. List full details of all flights from HROW (Heathrow), EMID (East Midlands) or BIRM (Birmingham) where the allocated aircraft is an DC9.

5. List details of aircraft with seat capacities between 50 and 120.

6. List the FlightNo's and service details of flights from HROW (Heathrow) to BIRM (Birmingham) with departure times after 07:00 and before 15:00.

7. List the names and addresses of all passengers whose names start with an initial of 'A' and have surnames of either 'Smith' or 'Smithson'.

4.3 Ordering the Rows of the Result.

The relational database model places no significance on the order in which rows appear in a table. As such the order in which rows will appear in a given query result should be considered to be indeterminate. Where a particular order is a required the ORDER BY clause must be used.

```
SELECT column-list

FROM table-list

WHERE conditional-expression

ORDER BY order-columns ;
```

When used the ORDER BY clause must appear as the last clause in the SELECT statement.

The column or columns on which the result is to be ordered must appear in the column-list of the SELECT clause.

Columns in the ORDER BY clause may be of type numeric, character or date.

4.3.1 Ordering on a Single Column.

Example 4.3.1 - List in descending order of FlightNo the Tickets (No's) and FlightNo's for Flights BD54, BD80, BD412, BD582, or BD332.

```
SELECT TicketNo, FlightNo, FlightDate

FROM Itinerary

WHERE FlightNo IN ('BD54','BD80','BD412', 'BD582', 'BD332')

ORDER BY FlightDate DESC ;
```

Result:

```
----------------------------------------------------------------------



ticketno  flight    flightdate

100100    BD54      02/10/94
100052    BD332     09/09/94
100051    BD582     07/09/94
100011    BD54      12/08/94
100022    BD412     07/08/94
100001    BD80      05/08/94
100002    BD80      05/08/94
100021    BD412     02/08/94
100041    BD412     02/08/94






----------------------------------------------------------------------
```

4.3.2 Ordering on Multiple Columns.

As well as being able to order the results of a query by a single column we can specify a list of columns on which we wish to order the result. The first column on the list provides the primary order, the second column is the ordered within the first and so on.

Example 4.3.2 - List, in ascending order of FlightDate within TicketNo - TicketNos, FlightNo's and FlightDates for TicketNos in the range 100010 to 100020.

```
SQL> SELECT TicketNo, FlightNo, FlightDate
```

```
2 FROM Itinerary

3 WHERE TicketNo BETWEEN 100010 AND 100020

4 ORDER BY TicketNo, FlightDate ;
```

Result:

```
----------------------------------------------------------------------



ticketno  flight    flightdate


100010    BD82      10/08/94
100011    BD54      12/08/94
100012    BD655     15/08/94
100012    BD776     15/08/94
100012    BD652     20/08/94
100012    BD775     20/08/94
100020    BD655     20/08/94
100020    BD772     20/08/94
100020    BD652     23/08/94





----------------------------------------------------------------------
```

Example 4.3.3 - List in descending order of FlightDate within ascending TicketNo, TicketNo's, FLightNo's and their FlightDates for Ticket Nos in the range 100010 to 100020.

```
SELECT TicketNo, FlightNo, FlightDate

FROM Itinerary

WHERE TicketNo BETWEEN 100010 AND 100020

ORDER BY TicketNo, FlightDate DESC ;
```

Result:

```
----------------------------------------------------------------------



ticketno  flight    flightdate


100010    BD82      10/08/94
```

```
100011   BD54     12/08/94
100012   BD652    20/08/94
100012   BD775    20/08/94
100012   BD655    15/08/94
100012   BD776    15/08/94
100020   BD652    23/08/94
100020   BD655    20/08/94
100020   BD772    20/08/94
```

```
----------------------------------------------------------------------
```

4.3.3 Ordering on Calculated (Result) Columns.

In addition to being able to order the result of a query by a given table column it is also possible to order by the values of an expression which appears within the SELECT clause, ie by a calculated column.

Example 4.3.4 - List the Tariffs available on Route No 9 (Birmingham-Brussels) which when VAT (17.5%) is added cost more than $170.00.

If we wished to Order the result of this query in descending order of the calculated VAT inclusive column the SQL would appear as follows.

```
SELECT FareType, Price*1.175 As VatPrice

FROM Tariff

WHERE RouteNo = 9

AND (Price*1.175) > 170

ORDER BY VatPrice DESC;
```

Result:

```
----------------------------------------------------------------------



faretype  vatprice

EUR       $212.68 PXR       $179.78




----------------------------------------------------------------------
```

Exercise 3.

Give the SQL required to:

1. Modify the query given for Example 4.3.1 so that the result is listed in ascending order by FlightNo.

2. List full details of flights from BIRM (Birmingham) to BRUS (Brussels) in descending order by DepTime.

3. List full details of all flights in ascending order of Service within descending order of Deptime.

4.4 Grouping the Rows of the Result.

We noted in section 4.1.4 that an aggregate (group) column could not be mixed in the SELECT clause with multi-valued columns. There are however, occasions when it is useful to be able to list several groups within a single query. To enable this the SELECT statement has an optional GROUP BY clause which is included immediately after the WHERE clause (or after the FROM clause if the WHERE clause is not present).

```
SELECT column-list

FROM table-list

WHERE conditional-expression

GROUP BY group-columns ;
```

4.4.1 Using the GROUP BY Clause.

Example 4.4.1 - Count the number of flights on the Itineraries of TicketNo's 100010 to100020 inclusive.

```
SELECT TicketNo, COUNT(FlightNo)

FROM Itinerary

WHERE TicketNo BETWEEN 100010 AND 100020

GROUP BY TicketNo ;
```

Result:

```
----------------------------------------------------------------------



ticketno  col2

100010    1
100011    1
100012    4
100020    3
```

4.4.2 Using the HAVING Clause.

In the same way that you can select specific rows through the use of the WHERE clause, you can also

select specific groups with the HAVING clause.

The HAVING clause compares some property of the group with a constant value. If a group fulfils the condition of the logical expression in the HAVING clause it is included in the result.

```
SELECT column-list

FROM table-list

WHERE conditional-expression

GROUP BY group-columns

HAVING group-conditional-expression ;
```

Example 4.4.2 - List TicketNo's with itineraries of 4 or more flights.

```
SELECT TicketNo, COUNT(FlightNo)

FROM Itinerary

GROUP BY TicketNo

HAVING COUNT(FlightNo) >= 4 ;
```

Result:

```
----------------------------------------------------------------------


ticketno col2


100012   4
100030   4
100100   6
```

Exercise 4.

Give the SQL required to:

1. The SQL given in Example 4.4.2 counts the FlightNo column. However, as there is one row for each flight in a given Ticket's itinerary it is not necessary for the SQL to specify any particular column. Modify the SQL given to reflect this fact.

2. List Routes (RouteNo) with less than 4 timetabled flights.

3. List the most popular FareType based on currently recorded Itineraries. Your query should list each FareType and its number of occurrences.

4.5 Joining Tables.

All of the queries considered up to this point have been based on single tables, ie resulting in one or more columns selected from a single table named within the FROM clause.

Where queries are to be based on the columns of two or more tables the required results are obtained by joining these tables together. The joined tables are specified in the FROM clause, and are "linked" to each other by one or more common columns; ie columns containing a range of values which each table has in common. The WHERE clause is then be used to specify the conditions between these common columns on which the required rows will be selected.

4.5.1 Equi-Joins.

Example 4.5.1 - Find the seating capacity of the aircraft allocated to flight BD80.

The type of aircraft (AircraftType) allocated to a particular flight (FlightNo) is recorded in the Flight table whilst the seating capacity for each type of aircraft is recorded in the Aircraft table.

Inspection of the Aircraft and Flight tables reveals that they are related to each other via the common column, AircraftType. Hence we can join the Aircraft table and the Flight table, selecting rows, where FlightNo in the Flight table is BD80 and AircraftType in the Flight table is equal to AircraftType in Aircraft table.

The following SQL illustrates the required join; known as an equi-join because the comparison operator in the join condition is = (equals).

```
SELECT NoSeats

FROM Aircraft, Flight

WHERE FlightNo = 'BD80'

AND Aircraft.AircraftType = Flight.AircraftType ;
```

Result:

```
noseat

300




--------------------------------------------------------------------
```

We are not restricted to joins involving only two tables here is an example of a query which needs to join three tables to obtain the required result.

Example 4.5.2 - List the description and seating capacity of the aircraft allocated to the flights on TicketNo 100010.

```
SELECT ADescription, NoSeats

FROM Aircraft, Flight, Itinerary

WHERE TicketNo = 100010

AND Itinerary.FlightNo = Flight.FlightNo

AND Flight.AircraftType = Aircraft.AircraftType;
```

Result:

```
--------------------------------------------------------------------


adescription      noseat

Boeing 737-300    300
Jet




--------------------------------------------------------------------
```

4.5.2 Table References and Table Aliases (Correlation Names).

When joined tables have a common column-name you must make it clear which table is being referenced by prefixing the column with its table-name. The column NoSeats in the previous examples is not prefixed because this column-name appears only in the Aircraft table and is therefore unambiguous.

The need to use a table-name prefix leads to fairly long column-name references. To avoid this, tables may be allocated an abbreviated name (an alias) within the FROM clause by which the table will be referenced elsewhere within the SELECT statement. The FROM clause then has the following general form

```
... FROM table-name alias, ... , table-name alias, ...
```

where the alias appears immediately after the table-name (separated by a space).

Using table aliases the previous example could be rewritten as:-

```
SELECT ADescription, NoSeats

FROM Aircraft A, Flight F, Itinerary I

WHERE TicketNo = 100010

AND I.FlightNo = F.FlightNo

AND F.AircraftType = A.AircraftType;
```

Result:

```
----------------------------------------------------------------------



adescription      noseat

Boeing 737-300    300
Jet





----------------------------------------------------------------------
```

4.5.3 Non-Equi-Joins.

Example 4.5.3 - A passenger requests details of alternative flights departing earlier than his/her currently booked Flight (BD659 18:25 Birmingham->Brussels).

The objective of this query is to find alternative flights which serve the same route as that served by BD659 with the same departure airport but with earlier departure times.

The conditional expression for the join used in this query uses the inequality operator > {greater than} and therefore represents a non-equi-join.

```
SELECT E.FromAirport, E.FlightNo, E.DepTime

FROM Flight F, Flight E
```

```
WHERE F.FlightNo = 'BD659'

AND F.FromAirport = E.FromAirport

AND F.RouteNo = E.RouteNo

AND F.DepTime > E.DepTime ;
```

Result:

```
---------------------------------------------------------------------



fromai   flight    deptime

BIRM     BD651     07:30
BIRM     BD655     15:00
BIRM     BD657     17:30






---------------------------------------------------------------------
```

4.5.4 Joining A Table to Itself.

Table aliases used with example 4.5.2 to abbreviate table-names also allow you to give a table a double alias thereby allowing you to join a table to itself as though it were two separate tables. This is a very useful facility where a query needs to compare rows within the same table. Example 4.5.3, as well as illustrating a non-equi-join also makes use of a self join with two aliases of the Flight table. This allows alternative flights to be found which serve the same route as the given (currently booked) flight.

Here is another example of the use of table aliases to enable a "self join".

Example 4.5.4 - To fly from Heathrow (HROW) to Brussels (BRUS) passengers must undertake their journey in two stages. List the FlightNo's, Airport's, Departure and Arrival times for inter-connecting flights from Heathrow to Brussels.

```
SELECT DISTINCT B.FlightNo, B.FromAirport,B.DepTime, B.ArrTime

FROM Flight A, Flight B

WHERE A.FromAirport = 'HROW'

AND A.ToAirport = B.FromAirport

AND B.ToAirport = 'BRUS';
```

Result:

```
----------------------------------------------------------------------

flight   fromai   deptime   arrtime

BD651    BIRM     07:30     09:35
BD655    BIRM     15:00     17:05
BD657    BIRM     17:30     19:35
BD659    BIRM     18:25     20:30

----------------------------------------------------------------------
```

Exercise 5.

Give the SQL required to:

1. List the full name of the airport from which flight BD275 departs.

2. List the full description and seating capacity of the aircraft allocated to flight BD582.

3. List the description and seating capacity of all the aircraft allocated to flights on the Itineraries of Tickets issue to 'R H Miller'.

4. List the FareTypes and Descriptions of the Fares available on RouteNo 4 (Heathrow - Edinburgh).

5. Calculate the total ticket price for Ticket number 100010 using Price from the Tariff table.

6. List the full name of the airport from which flight BD257 departs and the full name of the airport at which flight BD257 arrives.

(Hint each flight record refers to two Airport records therefore the Airport table should appear twice in the FROM clause. Use aliases to differentiate the first Airport reference from the second.)

7. List for the flights on all East-Midlands Routes the FlightNo, service provided, aircraft used, and seatingcapacity.

(Hint use LIKE to obtain RouteNo's and join Route and Flight tables on RouteNo. How are you going to find the seating capacity for each flight?)

4.6 Modifying the Contents of Database Tables.

SQL provides three statements with which you can modify the contents of database tables.

UPDATE which modifies data in existing rows of a table.

INSERT which inserts new rows into a table.

DELETE which deletes existing rows from a table.

If you only have SELECT privileges granted on the tables of the "airline" database the DBMS will prevent you from executing any of the UPDATE, INSERT, or DELETE examples given in this section.

To give you practice in the use of these statements, examples and exercises are provided at the end of Section 5 which will give you an opportunity to define and manipulate your own tables. As owner, of these tables you will automatically have select, insert, update and delete privileges.

4.6.1 Updating Rows.

The UPDATE statement consists of three clauses

```
UPDATE tablename

SET column-assignment-list

WHERE conditional-expression ;
```

where the column-assignment-list lists the columns to be updated and the values they are to be set to and takes the general form:

```
column-name = value, column-name = value, ...
```

where value may either be a constant or a column-expression which returns a value of the same type as column-name.

The WHERE clause is optional. When used, the WHERE clause specifies a condition for UPDATE to test when processing each row of the table. Only those rows which test True against the condition are updated with the new values.

Example 4.6.1 - Increase the price of the Standby Single (STS) Tariff on Route 13 (East Midlands-Paris) by $5.00.

```
UPDATE Tariff

SET Price = Price + 5

WHERE RouteNo = 13

AND FareType = 'STS' ;
```

Without the WHERE clause the UPDATE statement updates all rows of the table.

Example 4.6.2 - Increase the price of Tariffs on all Routes by 10 percent.

```
UPDATE Tariff

SET Price = Price * 1.1 ;
```

4.6.2 Inserting Rows.

The INSERT statement has two distinct variations the first, and simplest, inserts a single row into the named table.

INSERT INTO tablename [(column-list)]

VALUES (constant-list) ;

Example 4.6.3 - Let us assume that two new types of aircraft are to be brought into service by the airline company; the Shorts-360 and Fokker-Friendship F24.

The type code (AircraftType) for these aircraft will be 'S60' and 'F24' respectively. The 'S60' has a seating capacity of 36 and the 'F24' has a seating capacity of 48.

INSERT INTO Aircraft

VALUES ('S60', 'Shorts-360', 36);

INSERT INTO Aircraft

VALUES ('F24', 'Fokker-Friendship', 48);

(Note if the input values match the order and number of columns in the table then column-list can be omitted.)

4.6.3 Inserting Rows Copied from Another Table.

The INSERT statement may also be used in conjunction with a SELECT statement query to copy the rows of one table to another.

The general form of this variation of the INSERT statement is as follows:

INSERT INTO tablename [(column-list)]

SELECT column-list

FROM table-list

WHERE conditional-expression ;

where the SELECT statement replaces the VALUES clause.

Only the specified columns of those rows selected by the query are inserted into the named table.

The columns of the table being copied-from and those of the table being copied-to must be type compatible.

If the columns of both tables match in type and order then the column-list may be omitted from the INSERT clause.

Example 4.6.4 - Copy Passenger records with Pids in the range 91 - 94 to an archive table APassenger.

INSERT INTO APassenger

```
SELECT *

FROM Passenger

WHERE Pid BETWEEN 91 AND 94 ;
```

(Note the APassenger table must be in existence at the time this statement is executed.)

4.6.4 Deleting Rows.

The general form of the DELETE statement is

```
DELETE FROM tablename

WHERE conditional-expression
```

The DELETE statement removes those rows from a given table that satisfy the condition specified in the WHERE clause.

Example 4.6.5 - Delete rows from the Passenger table for Pid's 91, 92, 93, and 94.

```
DELETE

FROM Passenger

WHERE Pid IN (91, 92, 93, 94) ;
```

Example 4.6.6 - Remove all Archived passengers records from the 'live' Passenger table.

This example of DELETE employs a sub-query as a part of the WHERE clause to compile a list of Pid's from the "archived passenger table", APassenger.

Where rows in the Passenger table are found with a Pid value which matches a value in the list they are removed by the DELETE statement. (See section 6.1 on sub-queries later on.)

```
DELETE

FROM Passenger

WHERE Pid IN

( SELECT Pid

FROM APassenger ) ;
```

The WHERE clause is optional, when omitted all rows of the named table are removed. For example

```
DELETE

FROM Aircraft ;
```

would delete all rows from the aircraft table.

4.6.5 Using Rollback, Savepoint and Commit.

Table updates, deletions and insertions are not made permanent until you end your session and exit from the DBMS. Whilst your database tables are in this uncommitted state you can see the changes you have made as if they were permanent but other users who may have privileges to access your tables cannot. Further, at any time during the period that your tables are in an uncommitted state you may reinstate the changed tables to their state prior to those changes by using the rollback statement which is simply entered as:

```
rollback ;
```

In addition it is possible to limit how much of your work Ingres will rollback by issuing savepoints between the consecutive SQL statements of a given transaction. Savepoints are issued with a name (which may begin with a number) which is then used in conjunction with rollback to identify the point in the transaction to rollback to.

(A transaction commences on the execution of your first SQL statement and includes all subsequent SQL statements issued up to the point where you issue a commit or at the point where you end your session.) So for example:

```
INSERT INTO Aircraft

VALUES ('S60', 'Shorts-360', 36);

savepoint 1;

INSERT INTO Aircraft

VALUES ('F24', 'Fokker-Friendship', 48);

rollback 1; /*undoes the last insert only*/

rollback; /*undoes the whole transaction*/
```

At any time during an SQL session your uncommitted database changes may be committed to the database by issuing the commit statement as follows:

```
commit ;
```

Having committed your pending database changes all users will then be able to see those changes.

Committed changes cannot be undone.

<Image>

SQL: Chapter 5: SQL Data Definition Language

Database Administration   VIEWS. (Virtual Tables and Data Security)

5.1 Database Administration.

All of the statements considered in previous sections belong to the data manipulation part of the SQL language and allow users to interrogate and change the data of selected database tables.

In this section we will consider two statements which form the data definition part of the INGRES/SQL language, namely:

CREATE used to create table definitions

DROP used to remove unwanted tables from the database.

In a live database, ie a database supporting some aspect of an organisation's operation, these statements (and others beside) would be used by the database administrator (DBA) to establish and maintain the definition of the database tables which support the organisations information needs.

The following examples of the CREATE statement are shown with INGRES data types.

5.1.1 Creating Tables.

In it simplest form the SQL (DDL) statement used to create a database table has the following syntax:

```
CREATE TABLE table-name

( column-definition-list ) ;
```

Example 5.1.1 - Let us consider the CREATE statement used to create the Airport table definition for the Airline Database.

```
CREATE TABLE Airport

(airport char(4) not null,

aname varchar(20),

checkin varchar(50),

resvtns varchar(12),

flightinfo varchar(12) );
```

Table Name.(Airport)

The name chosen for a table must be a valid name for your DBMS. (See valid Ingres names.)

Column Names.(Airport, AName, ..., FlightInfo)

The names chosen for the columns of a table must also be a valid name for your DBMS. (See valid Ingres names.)

Data Types.

Each column must be allocated an appropriate data type - (See Ingres data types.)

In addition, key columns, ie columns used to uniquely identify individual rows of a given table, may be specified to be NOT NULL. The DBMS will then ensure that columns specified as NOT NULL always contain a value.

(Note zero and space are values.)

Valid Ingres Names.

The rules for naming INGRES objects (such as databases, tables, columns, views, forms, etc) are as follows:

Names can contain only alphanumeric characters and must begin with an alphabetic character or an underscore (_). Database names must begin with an alphabetic character and can not begin with an underscore. Case is not significant. Ingres converts all upper case letters to lowercase. Ingres object names can contain (but not begin with) any of the following special characters: "0" - "9", "#", "@", "$". Tables names cannot begin with "ii" or "sq"; these are reserved for use by Ingres.

The maximum length of an Ingres object name is 24 characters.

Ingres Data Types.

Notation	Type	Range
char(1) - char(2000)	character	A string of 1 to 2000 characters
c1 - c2000	character	A string of 1 to 2000 characters
varchar(1) - varchar(2000)	character	A string of 1 to 2000 characters
text(1) - text(2000)	character	A string of 1 to 2000 characters
integer1	1-byte integer	-128 to +127
smallint	2-byte integer	-32,768 to +32,767
integer	4-byte integer	-2,147.483,648 to +2,147,483,647

float4	4-byte floating	-1.0e+38 to +1.0e+38 (7 digit precision)
float	8-byte floating	-1.0e+38 to +1.0e+38 (16 digit precision)
date	date (12 bytes)	1-jan-1582 to 31-dec-2382 (for absolute dates) and -800 to +800 years (for time intervals)
money	money (8 bytes)	$-999,999,999,999.99 to $999,999,999,999.99
table_key	character	no range: stored as 8 bytes
object_key	character	no range: stored as 16 bytes

5.1.2 Copying Tables.

It is also possible with the following variation of the CREATE statement to create a new table definition and copy rows into the new table with a single statement.

```
CREATE TABLE table-name ( column-definition-list )

AS SELECT column-list

FROM table-list

WHERE conditional-expression ;
```

Example 5.1.2 - Let us suppose that want to create an exact copy of the Ticket table which we will call Ticket2.

```
CREATE TABLE Ticket2 (TicketNo, TicketDate, Pid)

AS SELECT TicketNo, TicketDate, Pid

FROM Ticket ;
```

As Ticket2 is inheriting the same number of columns, the same order of columns, and the same column names as Ticket, the column-lists used in this example could have been omitted.

If we had wanted to create Ticket2 containing only a sub-set of the rows from the Ticket table then we would have used the WHERE clause to specify the required selection conditions.

5.1.3 Full Ingres Create Table Syntax.

Create a new base table owned by the user who issues the statement:

create table tablename

(columnname format {, columnname format})

[with_clause]

To create a table and load from another table:

create table tablename

[(columnname {, columnname})] as subselect {union [all] subselect}

[with_clause]

Where the with_clause consists of the word with followed by a comma-separated list of one or more of the following:

structure = hash | heap | isam | btree /*default is heap */

location = (locationname {, locationname}) /*default is ii_database */

[no]journaling /* default without journaling */

[no]duplicates /* default duplicates allowed */

key = (columnlist)

fillfactor = n /* defaults - 50%hash,

80%isam, 80%btree */

minpages = n /* No of hash primary pages - default 16 */

maxpages = n /* No of hash primary pages - default no limit */

leaffill = n /* %fill - btree leaf index pages - default 70% */

nonleaffill = n /* %fill - tree nonleaf indexpages - default 80%*/

compression[= ([[no]key] [,[no]data])] | nocompression

A table can have a maximum of 300 columns and a table row may be a maximum of 2000 bytes wide. (Note nullable columns require an additional byte of storage for the null indicator.)

The name and data type of each column in the new table are specified by columnname and format arguments respectively.

Columnname can be any valid Ingres name - must me unique within the table.

Format specifies the datatype and length of the column and uses the following syntax:

data type [not null [with | not default] | with null] [not | with system_maintained]

with null (assumed if [not null] omitted)

The column accepts nulls. (Ingres inserts a null if no value supplied.)

not null with default

The column does not accept nulls.(Ingres supplies a default value if no value given.)

not null (= not null not default)

The user must supply a value.

with system_maintained

Used in conjunction with a logical key data types (ie table_key or object_key). Ingres automatically assigns a unique value to the column when a row is appended to the table.

Only compatible with not null with default.

System_maintained columns cannot be changed by users or applications.

5.1.4 Removing Unwanted Tables.

When a database table becomes redundant it may be dropped from the database as follows.

DROP TABLE Table2 ;

Exercise 6. (This exercise must be followed in sequence.)

Give the appropriate SQL to following:

1) Create the archive table APassenger with the same column names and data types as the Passenger table. Reference Appendix A6.2 or use the help statement to see the Passenger table definition.

2) Copy rows to the Apassenger table from the Passenger table according to the SQL specification given in Example 4.6.4. Use appropriate SQL to confirm that you have copied the required rows into the APassenger table.

3) Create using a single statement a new table called People which is an exact copy of (ie contains the same rows of data) the Passenger table.

4) Issue a COMMIT after creating the People table.

5) Delete those rows (passenger records) in the People table which appear in the Apassenger table. Use appropriate SQL to confirm that you have deleted the required records.

6) Issue a ROLLBACK. Use appropriate SQL to confirm the results.

7) Create using a single statement a new table called Craft which is an exact copy of (ie contains the same rows of data) as the Aircraft table.

8) Add the two new aircraft types given in Example 4.6.3 to the Craft table. Use appropriate SQL to confirm that the two new aircraft have been inserted correctly.

9) Reduce by 4 the seating capacity of all aircraft recorded in the Craft table. Use appropriate SQL to confirm that you have updated the table correctly.

10) KEEP the Craft table you will need it later! However, make a note of how you could have removed this table at this point without using the Drop statement.

Drop the APassenger and People tables created in this exercise.

5.2 VIEWS. (Virtual Tables and Data Security)

A View, as we will see, is a definition for a "virtual table" (virtual because there is no permanent allocation of storage space) which is assembled at reference time from selected rows and/or columns of one or more real tables.

A view may be queried in exactly the same way as a real table.

Views are useful for two main reasons:

1) they enable users to see data, from a generalised database design, in the form most convenient for their needs.

2) they may be employed as a security mechanism for restricting user access to specific tables columns and/or rows.

The statement used to create a view has the following general form:

```
CREATE VIEW view-name [( column-list )]

AS SELECT column-list

FROM table-list

WHERE conditional-expression ;
```

You can display the specification of the views you have created using the help statement whose general syntax is as follows:

```
help view view_name {,view_name}
```

When a view is no longer required it may be dropped from the database with the DROP statement:

```
DROP VIEW table-name ;
```

5.2.1 Views Designed to Simplify Queries.

As we noted with Example 4.5.4, there are no direct flights from Heathrow (HROW) to Brussels (BRUS). To simplify the query required to list the departure times of interconnecting flights we will specify a view called Brussels-Link.

```
CREATE VIEW Brussels_Link

AS SELECT DISTINCT B.FlightNo,B.FromAirport,

B.DepTime, B.ArrTime

FROM Flight A, Flight B

WHERE A.FromAirport = 'HROW'

AND A.ToAirport = B.FromAirport

AND B.ToAirport = 'BRUS' ;
```

Example 5.2.1 - List the FlightNo's, Airport's, Departure and Arrival times for flights from 1500 which link Heathrow with Brussels.

```
SELECT FromAirport, FlightNo, DepTime, ArrTime

FROM Brussels_Link

WHERE DepTime >= '15:00';
```

Result:

```
-----------------------------------------------------------------------



fromai   flightn  deptime   arrtime

BIRM     BD655    15:00     17:05
BIRM     BD657    17:30     19:35
BIRM     BD659    18:25     20:30






-----------------------------------------------------------------------
```

5.2.2 Views Providing Database Security.

There are many instances where access to private data within a database needs to be restricted to specific users.

It is possible to create such access restrictions using views. In this section we will consider just one example of how access may be controlled by creating a view which consists of different rows depending on the user who is querying the view.

Example 5.2.2 - Let us suppose that we want to allow all passengers to view their itineraries from a visual display at the airport by logging on to the Airline's DBMS under their Passenger ID as held in the Passenger table (the Pid column).

The DBMS provides a pseudo-column user which holds the name (login) of the person currently querying the database. The pseudo-column user may then be used in the WHERE clause like any other table column. Hence the following view will automatically return the itinerary belonging to the enquiring passenger.

```
CREATE VIEW PassengerItinerary

AS SELECT I.LegNo, I.TicketNo, I.FlightNo, I.FlightDate

FROM Passenger P, Itinerary I

WHERE P.Pid = user

AND P.TicketNo = I.TicketNo

ORDER BY TicketNo, LegNo ;
```

This view would need to be accessible to all, so we would need to assign read-only privileges to the view as follows. We would do this with the grant statement as follows:

```
grant select on PassengerItinerary to public;
```

The following SQL would then list only the itinerary belonging to the enquiring passenger.

```
SELECT * FROM PassengerItinerary ;
```

Exercise 7.

Give the SQL required to

1. Passengers travelling from Heathrow to Paris must pick up a link flight from East Midlands. Create a View of the interconnecting flights between HROW (Heathrow) and PARI (Paris).

2. List, based on an appropriate join with the view created in (1), the possible arrival times at Paris based upon a departure from Heathrow on flight BD224.

3. Remove the View created in (1) from the database.

<Image>

SQL: Chapter 6: Further Data Manipulation Techniques

Introduction Sub-Queries Correlated Sub-Queries Set Operators

----

6. Introduction

This section returns to the SELECT statement and considers some slightly more complex queries which demonstrate the flexibility and strength of SQL as a data manipulation language.

6.1 Sub-Queries.

Example 6.1.1 - List the description of the aircraft with the largest seating capacity.

This appears to be a very straight forward and simple query. But it requires a query which will need to compare every aircraft row with all other rows of the aircraft table to find the one row for the aircraft with the largest seating capacity.

The SQL needed to find the largest seating capacity (the largest value of NoSeats) is simply:

```
SELECT MAX(NoSeats)

FROM Aircraft;
```

Result:

```
col1

300
```

But to which AircraftType does this seating capacity belong? We need to list the AircraftType which corresponds to this value of NoSeats, ie

```
SELECT AircraftType

FROM Aircraft

WHERE NoSeats = "largest seating capacity"
```

SQL allows us to specify a sub-query within the WHERE clause which is executed before the main outer query to return one or more rows which may then be compared with the rows returned by the outer query. Hence putting the above queries together the required SQL is as follows.

```
SELECT AircraftType

FROM Aircraft

WHERE NoSeats = ( SELECT MAX(NoSeats)

FROM Aircraft );
```

Result:

```
----------------------------------------------------------------------


aircra

737




----------------------------------------------------------------------
```

Note the sub-query appears is in brackets indicating that it will return its result before the execution of the outer query.

This query could have been alternatively expressed as:

```
SELECT AircraftType

FROM Aircraft

WHERE NoSeats >= ALL ( SELECT NoSeats

FROM Aircraft );
```

Example 6.1.2 - List all aircraft which are not allocated to any timetabled Flights

For this example we will consider the SQL components required for each part of the query. Firstly we need to obtain a list of aircraft types from the Flights tables; those aircraft which are allocated to one or more timetabled flights, ie:

```
SELECT DISTINCT AircraftType

FROM Flights;
```

Result:

```
----------------------------------------------------------------------

aircra

ATP
DC9
737

----------------------------------------------------------------------
```

We require the AircraftTypes recorded in the Aircraft table which do not appear in this list,ie:

(Craft table created earlier will be used for this example.)

```
SELECT AircraftType

FROM Craft

WHERE AircraftType NOT IN ("the list of allocated aircraft")
```

Putting these two queries together as before gives:

```
SELECT AircraftType

FROM Craft

WHERE AircraftType NOT IN

( SELECT DISTINCT AircraftType

FROM Flights );
```

Result:

```
----------------------------------------------------------------------

aircra

S60
F24
```

Exercise 8.

1. By use of the IN operator and a sub-query list the names and addresses of passengers with tickets for flight BD80.

2. List the names and addresses of passengers with Standby Single (SBS) or Standard Return (SDR) tickets for flight BD54.

3. Provide an alternative to the SQL used in Example 6.1.2.

4. Find the AircraftType with the smallest seating capacity:

a) using an appropriate aggregate function;

b) without using an aggregate function.

6.2 Correlated Subqueries.

In the previous examples of subqueries each subquery was seen to executed once; returning its result for use by the main (outer) query.

In correlated sub-queries the sub-query executes once for each row returned by the outer query.

Example 6.2.1 - List the names and addresses of passengers with tickets made up of itineraries with only one flight.

```
SELECT Name, Address

FROM Passenger P

WHERE 1 = ( SELECT COUNT(*)

FROM Ticket T, Itinerary I

WHERE P.Pid = T.Pid AND

T.TicketNo = I.TicketNo );
```

Result:

```
name       address

A Smithson  16 Bedford St
C Evans     63 Kew Green
T Pittman   The Little House
```

```
K E        11 Rosedale Avenue
Kendall
```

```
---------------------------------------------------------------------
```

For each row in the passenger table the value of Pid for that row is passed into the subquery. The subquery executes once for each value of Pid passed from the outer query and returns the value of COUNT (the number of rows in the ticket-itinerary join, ie flight legs, which satisfy the subquery WHERE condition, ie flights for this Pid) to the WHERE clause in the outer query. Where COUNT equals one for the current Pid its corresponding row in the passenger table is listed in the result.

6.2.1 Testing for Existence.

The EXISTS operator enables us to test whether or not a subquery has returned any rows.

EXISTS evaluates to true if one or more rows have been selected and evaluates to false if zero rows have been selected.

NOT EXISTS evaluates to true if zero rows have been selected and evaluates to false if one or more rows have been selected.

Example 6.2.2 - List the names and addresses of passengers who purchased their Tickets on 01-AUG-92.

```
SELECT Name, Address

FROM Passenger P

WHERE EXISTS ( SELECT *

FROM Ticket T

WHERE P.Pid = T.Pid AND

T.TicketDate = '01/08/94 );
```

Result:

```
---------------------------------------------------------------------



name       address

J Millar    Englewood Cliffs
T Pittman   The Little House
```

6.3 Set Operators.

We have seen how a query may be composed of subqueries whose results are returned for evaluation within a WHERE clause.

We can also combine the results of pairs of queries and/or subqueries by using the following Set Operators.

MINUS returns query difference; ie all rows in the result of the first query which do not appear in the second query.

INTERSECTION returns common rows; ie only those rows which appear in the results of both the first query and the second query.

UNION combines query results; ie returns the distinct rows returned by either the first query or the second query.

It is important to note that queries to be operated on by these operators must be union compatible, ie the SELECT clauses of both queries must contain the same number and type of columns.

(Note MINUS and INTERSECTION are included here for completeness they are NOT available in Ingres.)

6.3.1 Query Difference. (Minus not available in Ingres)

The MINUS operator takes as its operands, the results of two separate queries and returns (as its result) the rows from the first query which do not appear in the results of second query.

Example 6.3.1 - Find names of the airports from which direct flights may be taken to all of the following destinations:

BELF (Belfast)

BIRM (Birmingham)

EDIN (Edinburgh)

EMID (East Midlands)

LBDR (Leeds/Bradford)

LVPL (Liverpool)

TEES (Teeside)

SELECT AName

FROM Airport A

```
WHERE NOT EXISTS

( ( SELECT DISTINCT F.ToAirport

FROM Flight F

WHERE F.ToAirport IN ('BELF','BIRM','EDIN',

'EMID','LBDR','LVPL','TEES') )

MINUS

( SELECT DISTINCT T.ToAirport

FROM Flight T

WHERE A.Airport = T.FromAirport ) );
```

This query would return Heathrow as the only airport.

The first subquery returns all flight destinations (airports) which exist in the Flights table corresponding with the given list.

The second subquery (correlated) takes each airport in turn from the Airport table (outer query) as a departure point (FromAirport) and generates a list of all possible direct destinations (ToAirport).

The MINUS operator compares the set of available destination airports with the set of direct destinations possible from each departure point (Airport). If there are no airports in the first query which are not in the second, then we have found a departure point from which we can reach all of the available destinations directly.

If there are zero rows returned, ie the list of available destinations not reached is zero, then the NOT EXISTS evaluates to true and the current ANAME from the Airport table is listed in the result.

6.3.2 Common Rows. (Intersection not available in Ingres)

The INTERSECT operator allows us to find the common rows from the results of two separate queries (or subqueries). The following example shows the intersection of two queries.

Example 6.3.2 - Find the Names and Addresses of passengers flying from HROW (Heathrow) to BIRM (Birmingham) and from BIRM to HROW. (Not necessarily on the same ticket.)

```
SELECT Name, Address

FROM Passenger P, Ticket T, Itinerary I, Flight F

WHERE P.Pid = T.Pid AND

T.TicketNo = I.TicketNo AND

I.FlightNo = F.FlightNo AND
```

```
F.FromAirport = 'HROW' AND

F.ToAirport = 'BIRM'

INTERSECT

SELECT Name, Address

FROM Passenger P, Ticket T, Itinerary I, Flight F

WHERE P.Pid = T.Pid AND

T.TicketNo = I.TicketNo AND

I.FlightNo = F.FlightNo AND

F.FromAirport = 'BIRM' AND

F.ToAirport = 'HROW' ;
```

Result would be:

```
NAME ADDRESS

----------- ---------------------------

G B Davis 25 Allenby Road

R H Miller 155 Kingston Road
```

The first query returns the names and addresses of passengers flying from Heathrow to Birmingham.

The second query returns the names and addresses of passengers flying from Birmingham to Heathrow.

The INTERSECT operator returns the rows from the first query which also appear in the rows from the second query, ie those passengers flying from Heathrow to Birmingham and from Birmingham to Heathrow.

6.3.3 Combining Results.

The UNION operator allows us to combine distinct rows from the results of two separate queries (or subqueries) The following example shows the union of two queries.

Example 6.3.3 - Find the Names and Addresses of passengers flying from HROW (Heathrow) to BIRM (Birmingham) and from BIRM to HROW or both.

```
SELECT Name, Address

FROM Passenger P, Ticket T, Itinerary I, Flight F

WHERE P.Pid = T.Pid AND

T.TicketNo = I.TicketNo AND
```

```
I.FlightNo = F.FlightNo AND

F.FromAirport = 'HROW' AND

F.ToAirport = 'BIRM'

UNION

SELECT Name, Address

FROM Passenger P, Ticket T, Itinerary I, Flight F

WHERE P.Pid = T.Pid AND

T.TicketNo = I.TicketNo AND

I.FlightNo = F.FlightNo AND

F.FromAirport = 'BIRM' AND

F.ToAirport = 'HROW' ;
```

Result:

```
-----------------------------------------------------------------------


name        address

D Etheridge 4 Maylands Avenue
G B Davis   25 Allenby Road
R H Miller  155 Kingston Road






-----------------------------------------------------------------------
```

The first query returns the names and addresses of passengers flying from Heathrow to Birmingham.

The second query returns the names and addresses of passengers flying from Birmingham to Heathrow.

The UNION operator returns the rows from the first query plus the rows from the second query and removes any resulting duplicates. Hence we see listed those passengers who are either flying from Heathrow to Birmingham, or who are flying from Birmingham to Heathrow, or both.

Exercise 9.

Give the SQL required to:

1. List the names and addresses of any passenger who on a single ticket have more than 5 flights.

2. Find the names of those passengers who are taking all of the flights which

J Millar is taking.

3. Rewrite the SQL statement given for Example 6.3.2 so that it compares flights for the same ticket and hence find those passenger's with return tickets HROW to BIRM and back. (Note. You will need to use correlated sub-queries and an alternative query strategy to find the intersection.)

4. Using correlated sub-queries find the Names of those passengers with a ticket to fly from HROW (Heathrow) to PARI (Paris) without a return flight to HROW; ie those passengers with non-return tickets.

<Image>

SQL: Appendix A: The Airline Database

Database Schema Table Specifications Table Data

Database Schema

Airport(Airport, AName, CheckIN, Resvtns, FlghtInfo)

Route(RouteNo, RDescription)

Fares(FareType, FDescription)

Tariff(RouteNo, FareType, Price)

Aircraft(AircraftType, ADescription, NoSeats)

Flight(FlightNo, FromAirport, ToAirport, DepTime, ArrTime, Service, AircraftType, RouteNo)

Passenger(Pid, Name, Address, TelNo)

Ticket(TicketNo, TicketDate, Pid)

Itinerary(TicketNo, FlightNo, LegNo, FlightDate, FareType)

Table Specifications

CREATE TABLE Airport

(airport char(4) not null,

```
aname varchar(20),
checkin varchar(50),
resvtns varchar(12),
flightinfo varchar(12) );
CREATE TABLE Route
( routeno integer1 not null,
rdescription varchar(40) );
CREATE TABLE Fares
(faretype char(3) not null,
fdescription varchar(25),
conditions varchar(40) );
CREATE TABLE Tariff
(routeno integer1 not null,
faretype char(3) not null,
price money );
CREATE TABLE Aircraft
( aircrafttype char(3) not null,
adescription varchar(40),
noseats smallint );
CREATE TABLE Flight
( flightno varchar(5) not null,
fromairport char(4),
toairport char(4),
deptime char(5),
arrtime char(5),
service varchar(20),
aircrafttype char(3),
routeno integer1 );
CREATE TABLE Passenger
(pid integer not null,
name varchar(20),
address varchar(40),
telno varchar(12) );
CREATE TABLE Ticket
(ticketno integer not null,
ticketdate date,
pid integer );
CREATE TABLE Itinerary
(ticketno integer not null,
flightno varchar(5) not null,
legno integer1 not null,
flightdate date,
faretype char(3) ) ;
```

Table Data

AIRPORT:

```
AIRPORT ANAME CHECKIN RESVTNS FLIGHTINFO
------------- -------------- ------------------------------------------------------- ------------ -----------------
AMST Amsterdam South Hall Departures 20 mins before flight 06 022 2426 20 178299
BELF Belfast Desks 18 and 19, 15 mins before flight 023 2325151 08494 22888
BIRM Birmingham Check-in desks 20 mins before flight 021 236 0121
BRUS Brussels Sabena check-in desks 30 mins before flight 2 511 9030 2 7207167
EMID East Midlands Check-in 15 mins before flight for heavy bags 0332 810552 0332 852614
DUBL Dublin Check-in 20 mins prior to departure 01 423 555
EDIN Edinburgh Gate 1 at least 10 mins before departure 031 447 1000
GLAS Glasgow Desks 60-64/Gate 8 20 mins before departure 041 204 2436
```

```
HROW Heathrow Island A/B Terminal-1 20 mins before flight 081 5895599 081 7457321
LBDR Leeds/Bradford Check-in 15 mins before flight for baggage 0532 451991
LVPL Liverpool 15 mins heavy baggages,10 mins hand baggage 051 494 0200
PARI Paris Hall 22 Terminal 1 30 mins before flight 14742 14444 14862 2280
TEES Teeside 15 mins heavy baggages,10 mins hand baggage 0642 219444
```

ROUTE:

```
ROUTENO RDESCRIPTION

------------- ---------------------

3 Heathrow-Belfast

4 Heathrow-Edinburgh

6 Heathrow-Leeds/Bradford

7 Heathrow-Liverpool

8 Heathrow-Teeside

9 Birmingham-Brussels

11 East Midlands-Belfast

13 East Midlands-Paris

14 Heathrow-Birmingham

15 Heathrow-East Midlands
```

FARES:

```
FARETYPE FDESCRIPTION CONDITIONS

-------------- ------------------- ---------------------------------

BUR Business Return Business use only

SDS Standard Single

SDR Standard Return

EUR Eurobudget Return Available Paris Brussels Amsterdam

KFS Key Fare Single

SBS Standby Single

EXR Excursion Return Same day return
```

PXR Super Key Return

APR Advanced Purchase Return 60 days advanced booking

SXR Superpex Return 90 days advanced booking

TARIFF:

```
ROUTENO FARETYPE PRICE

-------------- -------------- --------

3 BUR 117

3 SDR 158

3 SDS 79

4 SDR 162

4 SBS 49

6 BUR 117

6 SBS 42

6 KFS 53

7 SDR 128

8 SDS 74

9 PXR 153

9 EUR 181

9 APR 95

11 KFS 59

13 EXR 121

14 SDR 110

14 SBS 33

15 SBS 33
```

FLIGHT:

```
FLIGHTNO FROMAIRPORT TOAIRPORT DEPTIME ARRTIME SERVICE AIRCRAFTTYPE ROUTENO

------------- ------------------ -------------- ------------ ------------ --------------- -------------------- --------------
```

BD80 HROW BELF 0725 0840 Breakfast 737 3

BD82 HROW BELF 0930 1045 Breakfast 737 3

BD91 BELF HROW 1730 1840 Dinner 737 3

BD95 BELF HROW 2120 2230 Dinner 737 3

BD54 ROW EDIN 1040 1155 Breakfast 737 4

BD51 EDIN HROW 0710 0825 Breakfast 737 4

BD412 HROW LBDR 0850 0945 Breakfast DC9 6

BD414 HROW LBDR 1145 1235 Light Meal DC9 6

BD413 LBDR HROW 1020 1115 Light Meal DC9 6

BD419 LBDR HROW 1845 1940 Dinner DC9 6

BD582 HROW LVPL 0810 0900 Breakfast DC9 7

BD589 LVPL HROW 1925 2015 Dinner DC9 7

BD332 HROW TEES 0835 0935 Breakfast DC9 8

BD651 BIRM BRUS 0730 0935 Breakfast DC9 9

BD655 BIRM BRUS 1500 1705 Afternoon Tea DC9 9

BD657 BIRM BRUS 1730 1935 Dinner DC9 9

BD659 BIRM BRUS 1825 2030 Dinner DC9 9

BD652 BRUS BIRM 1005 1010 Breakfast DC9 9

BD656 BRUS BIRM 1750 1755 Afternoon Tea DC9 9

BD658 BRUS BIRM 2005 2010 Dinner DC9 9

BD660 BRUS BIRM 2100 2105 Dinner DC9 9

BD275 EMID BELF 1800 1855 Dinner DC9 11

BD255 EMID PARI 1250 1455 Lunch DC9 13

BD257 EMID PARI 1530 1735 Afternoon Tea DC9 13

BD256 PARI EMID 1530 1540 Afternoon Tea DC9 13

BD258 PARI EMID 1810 1820 Dinner DC9 13

```
BD772 HROW BIRM 0810 0900 Coffee ATP 14

BD774 HROW BIRM 1045 1130 Coffee ATP 14

BD776 HROW BIRM 1230 1320 Coffee ATP 14

BD778 HROW BIRM 1310 1355 Coffee ATP 14

BD780 HROW BIRM 1530 1625 Coffee ATP 14

BD782 HROW BIRM 1755 1840 Coffee ATP 14

BD771 BIRM HROW 0655 0745 Breakfast ATP 14

BD773 BIRM HROW 0930 1015 Coffee ATP 14

BD775 BIRM HROW 1200 1245 Coffee ATP 14

BD777 BIRM HROW 1420 1505 Coffee ATP 14

BD779 BIRM HROW 1640 1725 Coffee ATP 14

BD781 BIRM HROW 2010 2100 Coffee ATP 14

BD222 HROW EMID 0755 0845 Coffee ATP 15

BD224 HROW EMID 1100 1150 Coffee ATP 15

BD226 HROW EMID 1415 1505 Coffee ATP 15

BD228 HROW EMID 1655 1745 Coffee ATP 15

BD230 HROW EMID 1945 2035 Coffee ATP 15

BD221 EMID HROW 0645 0725 Breakfast ATP 15

BD223 EMID HROW 1235 1325 Coffee ATP 15

BD225 EMID HROW 1235 1325 Coffee ATP 15

BD227 EMID HROW 1535 1625 Coffee ATP 15

BD229 EMID HROW 1805 1855 Coffee ATP 15

AIRCRAFT:

AIRCRAFTTYPE ADESCRIPTION NOSEATS

-------------------- --------------------- -------------

ATP Advanced Turbo Prop 48

DC9 McDonnel Douglas Jet 120
```

```
737 Boeing 737-300 Jet 300

PASSENGER:

PID NAME ADDRESS Tel No

---- --------------- ------------------------------- ----------------

26 J Millar Englewood Cliffs 061 343 881

28 J D Ullman 1 Microsoft Way

29 A Smithson 16 Bedford St 071 577 890

30 D Etheridge 4 Maylands Avenue

34 E Simon 8 Cherry Street

10 D N Hamer 1 St Paul's Churchyard

20 D E Avison 5 Chancery Lane

21 G B Davis 25 Allenby Road

24 C Evans 63 Kew Green

90 A N Smith 81 Digby Crescent 071 321 456

91 T Pittman The Little House

92 J Peters 31 Lucas Road

93 K E Kendall 11 Rosedale Avenue

94 R H Miller 155 Kingston Road 0638 4672

TICKET:

TICKETNO TICKETDATE PID

-------------- ---------- ---

100001 01/08/94 26

100002 25/07/94 28

100010 09/08/94 29

100011 11/08/94 24

100012 01/06/94 21
```

```
100020 15/08/94 30
100021 28/07/94 34
100022 05/07/94 20
100030 30/08/94 94
100041 01/08/94 91
100051 01/09/94 92
100052 01/09/94 93
100100 15/09/94 94
ITINERARY:
TICKETNO FLIGHTNO LEGNO FLIGHTDATE FARETYPE
--------------- -------------- ---------- ------------------ ---------------
100001 BD80 1 05/08/94 BUR
100001 BD95 2 05/08/94 BUR
100002 BD80 1 05/08/94 SDR
100002 BD95 2 10/08/94 SDR
100010 BD82 1 10/08/94 SDS
100011 BD54 1 12/08/94 SBS
100012 BD776 1 15/08/94 SDR
100012 BD655 2 15/08/94 APR
100012 BD652 3 20/08/94 APR
100012 BD775 4 20/08/94 SDR
100020 BD772 1 20/08/94 SBS
100020 BD655 2 20/08/94 EUR
100020 BD652 3 23/08/94 EUR
100021 BD412 1 02/08/94 SBS
100021 BD419 2 07/08/94 SBS
100022 BD412 1 07/08/94 BUR
```

100022 BD419 2 08/08/94 BUR

100030 BD224 1 01/09/94 SBS

100030 BD255 2 02/09/94 EXR

100030 BD256 3 06/09/94 EXR

100030 BD275 4 06/09/94 KFS

100041 BD412 1 02/08/94 KFS

100051 BD582 1 07/09/94 SDR

100051 BD589 2 07/09/94 SDR

100052 BD332 1 09/09/94 SDS

100100 BD51 1 24/09/94 SDR

100100 BD774 2 24/09/94 SDR

100100 BD659 3 25/09/94 PXR

100100 BD658 4 01/10/94 PXR

100100 BD771 5 02/10/94 SDR

100100 BD54 6 02/10/94 SDR