POSTGRESQL is an open-source, full-featured relational database. This presentation gives an overview of how POSTGRESQL processes queries.
SELECT firstname
FROM friend
WHERE age = 33;
test=> \textbf{SELECT} \textit{firstname} \\
\texttt{test-> FROM friend} \\
\texttt{test-> WHERE age = 33;} \\
\textit{firstname} \\
\underline{-------------} \\
\textit{Sandy} \\
\texttt{(1 row)}
test=> SELECT firstname
test-> FROM friend
test-> WHERE age = 33;
[ query is processed ]

firstname

-------------------
Sandy
(1 row)
test=> SELECT firstname
    test->  FROM friend
    test->  WHERE age = 33;

Breakpoint 1, PQexec (conn=0x807a000,
    query=0x8081200  "SELECT firstname\nFROM friend\nWHERE age = 33;")
    at fe-exec.c:1195
Libpq
TCP/IP Packet

ack 61 win 8760 <nop,nop,timestamp 137847 7276138> (DF)

0000: 00 d0 b7 b9 b6 c8 00 02   b3 04 09 dd 08 00 45 00  ________ ______E_
0010: 00 62 45 31 40 00 40 06   b1 fe ac 14 00 02 a2 21  _bE1@_ _______!
0020: f5 2e c0 0d 15 38 1c af   94 34 a8 1a 1e 39 80 18  _.____ 4___9_
0030: 22 38 19 d5 00 00 01 01   08 0a 00 02 1a 77 00 6f  "8______ _____w_o
0040: 06 6a 51 53 45 4c 45 43   54 20 66 72 73 74 6e  ame_FROM firstn
0050: 61 6d 65 0a 46 52 4f 4d   20 66 72 69 65 6e 64 0a  _jQSELEC T friend_
0060: 57 48 45 52 45 20 61 67   3d 20 33 33 3b 00  WHERE ag e = 33;_
FindExec: found "/var/local/postgres/.bin/postgres" using argv[0]
DEBUG: connection: host=[local] user=postgres database=test
DEBUG: InitPostgres
DEBUG: StartTransactionCommand
DEBUG: query: SELECT firstname
       FROM friend
       WHERE age = 33;

[ query is processed ]

DEBUG: ProcessQuery
DEBUG: CommitTransactionCommand
DEBUG: proc_exit(0)
DEBUG: shmem_exit(0)
DEBUG: exit(0)
Query Processing

FindExec: found "/var/local/postgres/.bin/postmaster" using argv[0]
./bin/postmaster: BackendStartup: pid 3320 user postgres db test socket 5
./bin/postmaster child[3320]: starting with (postgres -d99 -P -d99 -v11072 -p test )
FindExec: found "/var/local/postgres/.bin/postmaster" using argv[0]
DEBUG: connection: host=[local] user=postgres database=test
DEBUG: InitPostgres
DEBUG: StartTransactionCommand
DEBUG: query: SELECT firstname FROM friend WHERE age = 0


DEBUG: rewritten parse tree:


DEBUG: emitting
DEBUG: ProcessQuery
DEBUG: CommitTransactionCommand
DEBUG: proc_exit(0)
DEBUG: shmem_exit(0)
DEBUG: exit(0)
./bin/postmaster: CleanupProc: pid 3320 exited with status 0
FindExec: found "/var/local/postgres/.bin/postgres" using argv[0]
DEBUG: connection: host=[local] user=postgres database=test
DEBUG: InitPostgres
DEBUG: StartTransactionCommand
DEBUG: query:
  SELECT firstname
  FROM friend
  WHERE age = 33;
DEBUG: parse tree:
  { QUERY
    :command 1
    :utility <>
    :resultRelation 0
    :into <>
    :isPortal false
    :isBinary false
    :isTemp false
    :hasAggs false
    :hasSubLinks false
    :table (
      { RTE
        :relname friend
        :relid 26912
        :subquery <>
        :alias <>
        :eref
          { ATTR
            :relname friend
            :attribs ("firstname" "lastname" "city" "state" "age")
          }
    
    :inh true
    :inFromC1 true
    :checkForRead true
    :checkForWrite false
    :checkAsUser 0
  )
Backend Flowchart — Magnified

1. Parse Statement
2. Traffic Cop
3. Utility Command
4. Generate Paths
5. Optimal Path
6. Generate Plan
7. Plan
8. Execute Plan
9. Rewrite Query
10. Utility
11. e.g. CREATE TABLE, COPY
12. SELECT, INSERT, UPDATE, DELETE, MERGE
Identifier Rule

identifier      {letter}{letter_or_digit}*

{identifier}    {

    int i;
    ScanKeyword *keyword;

    for(i = 0; yytext[i]; i++)
        if (isupper((unsigned char) yytext[i]))
            yytext[i] = tolower((unsigned char) yytext[i]);
    if (i >= NAMEDATALEN)
        {elog(NOTICE, "identifier \"%s\" will be truncated to \"%.s\", yytext, NAMEDATALEN-1, yytext);
            yytext[NAMEDATALEN-1] = '\0';
        }
    keyword = ScanKeywordLookup((char*)yytext);
    if (keyword != NULL) {
        return keyword->value;
    }
    else {
        yylval.str = pstrdup((char*)yytext);
        return IDENT;
    }

}
digit          [0-9]
letter         [\000-\377_A-Za-z]
letter_or_digit [\000-\377_A-Za-z0-9]

integer        {(digit)+
decimal        ((({digit}*.{digit}+) | ({digit}+\.{digit}*))
real           ((({digit}*.{digit}+) | ({digit}+\.{digit}*) | ({digit}+))([Ee][−+]?{digit}+)

{integer}       {
    char* endptr;
    errno = 0;
    yylval.ival = strtol((char*)yytext, &endptr, 10);
    if (*endptr != ‘\0’ || errno == ERANGE)
    {
        yylval.str = pstrdup((char*)yytext);
        return FCONST;
    }
    return ICONST;
}
{decimal}       {
    yylval.str = pstrdup((char*)yytext);
    return FCONST;
}
{real}          {
    yylval.str = pstrdup((char*)yytext);
    return FCONST;
}
SELECT "firstname" FROM "friend"
WHERE "age" = 33

--EOF (start condition 0)
simple_select: SELECT opt_distinct target_list
into_clause from_clause where_clause
group_clause having_clause
{
    SelectStmt *n = makeNode(SelectStmt);
    n->distinctClause = $2;
    n->targetList = $3;
    n->istemp = (bool) ((Value *) lfirst($4))->val.ival;
    n->into = (char *) lnext($4);
    n->fromClause = $5;
    n->whereClause = $6;
    n->groupClause = $7;
    n->havingClause = $8;
    $$ = (Node *)n;
}
typedef struct SelectStmt {
    NodeTag type;
    /* These fields are used only in "leaf" SelectStmts.
     */
    List *distinctClause; /* NULL, list of DISTINCT ON exprs, or
                        * lcons(NIL,NIL) for all (SELECT
                        * DISTINCT) */
    char *into; /* name of table (for select into table) */
    bool istemp; /* into is a temp table? */
    List *targetList; /* the target list (of ResTarget) */
    List *fromClause; /* the FROM clause */
    Node *whereClause; /* WHERE qualification */
    List *groupClause; /* GROUP BY clauses */
    Node *havingClause; /* HAVING conditional-expression */
    /* These fields are used in both "leaf" SelectStmts and upper-level
     * SelectStmts.  portalname/binary may only be set at the top level.
     */
    List *sortClause; /* sort clause (a list of SortGroupBy’s) */
    char *portalname; /* the portal (cursor) to create */
    bool binary; /* a binary (internal) portal? */
    Node *limitOffset; /* # of result tuples to skip */
    Node *limitCount; /* # of result tuples to return */
    List *forUpdate; /* FOR UPDATE clause */
    /* These fields are used only in upper-level SelectStmts.
     */
    SetOperation op; /* type of set op */
    bool all; /* ALL specified? */
    struct SelectStmt *larg; /* left child */
    struct SelectStmt *rarg; /* right child */
    /* Eventually add fields for CORRESPONDING spec here */
} SelectStmt;
Starting parse
Entering state 0
Reading a token: Next token is 377 (SELECT)
Shifting token 377 (SELECT), Entering state 15
Reading a token: Next token is 514 (IDENT)
Reducing via rule 534 (line 3430), \rightarrow \text{opt\_distinct}
state stack now 0 15
Entering state 324
Next token is 514 (IDENT)
Shifting token 514 (IDENT), Entering state 496
Reading a token: Next token is 314 (FROM)
Reducing via rule 871 (line 5391), IDENT \rightarrow \text{ColId}
state stack now 0 15 324
Entering state 531
Next token is 314 (FROM)
Reducing via rule 789 (line 4951), \rightarrow \text{opt\_indirection}
state stack now 0 15 324 531
Entering state 755
Next token is 314 (FROM)
Reducing via rule 760 (line 4591), ColId opt_indirection \rightarrow \text{c\_expr}
state stack now 0 15 324
Entering state 520
Reducing via rule 693 (line 4272), \text{c\_expr} \rightarrow \text{a\_expr}
state stack now 0 15 324
Entering state 519
Next token is 314 (FROM)
Reducing via rule 833 (line 5183), \text{a\_expr} \rightarrow \text{target\_el}
state stack now 0 15 324
Entering state 524
Reducing via rule 831 (line 5171), \text{target\_el} \rightarrow \text{target\_list}
state stack now 0 15 324
Entering state 523
Next token is 314 (FROM)
Reducing via rule 518 (line 3382), \rightarrow \text{into\_clause}
Scanning and Parsing

Starting parse
Entering state 0
Reading a token:
--(end of buffer or a NUL)
--accepting rule at line 476 ("SELECT")
Next token is 377 (SELECT)
Shifting token 377 (SELECT), Entering state 15
Reading a token:
--accepting rule at line 254 (" ")
--accepting rule at line 476 ("firstname")
Next token is 514 (IDENT)
Reducing via rule 534 (line 3430), \rightarrow opt_distinct
state stack now 0 15
Entering state 324
Next token is 514 (IDENT)
Shifting token 514 (IDENT), Entering state 496
Reading a token:
--accepting rule at line 254 ("\n")
--accepting rule at line 476 ("FROM")
Next token is 314 (FROM)
Reducing via rule 871 (line 5391), IDENT \rightarrow ColId
state stack now 0 15 324
Entering state 531
Next token is 314 (FROM)
Reducing via rule 789 (line 4951), \rightarrow opt_indirection
state stack now 0 15 324 531
Entering state 755
Next token is 314 (FROM)
typedef struct List {
    NodeTag type;
union {
    void *ptr_value;
    int int_value;
    } elem;
struct List *next;
} List;

#define NIL ((List *) NULL)
#define lfirst(l) ((l)->elem.ptr_value)
#define lnext(l) ((l)->next)
#define lsecond(l) lfirst(lnext(l))
#define lfirsti(l) ((l)->elem.int_value)
#define foreach(_elt_, _list_) \
for(_elt_=(list_); _elt_!=NIL; _elt_=lnext(_elt_))
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lfirst</td>
<td>returns value stored in List</td>
</tr>
<tr>
<td>lnext</td>
<td>returns pointer to next in List</td>
</tr>
<tr>
<td>foreach</td>
<td>loops through List</td>
</tr>
<tr>
<td>length</td>
<td>returns length of List</td>
</tr>
<tr>
<td>nth</td>
<td>returns nth element from List</td>
</tr>
<tr>
<td>makeList1</td>
<td>creates a new list</td>
</tr>
<tr>
<td>lcons</td>
<td>adds value to front of List</td>
</tr>
<tr>
<td>lappend</td>
<td>appends value to end of List</td>
</tr>
<tr>
<td>nconc</td>
<td>concatenates two Lists</td>
</tr>
</tbody>
</table>

There are versions of these functions for storing integers rather than pointers.
typedef struct RangeTblEntry {
    NodeTag     type;
    /*
     * Fields valid for a plain relation RTE (else NULL/zero):
     */
    char        *relname;     /* real name of the relation */
    Oid         relid;        /* OID of the relation */
    /*
     * Fields valid for a subquery RTE (else NULL):
     */
    Query       *subquery;    /* the sub-query */
    /*
     * Fields valid in all RTEs:
     */
    Attr        *alias;       /* user-written alias clause, if any */
    Attr        *eref;        /* expanded reference names */
    bool        inh;          /* inheritance requested? */
    bool        inFromCl;     /* present in FROM clause */
    bool        checkForRead; /* check rel for read access */
    bool        checkForWrite; /* check rel for write access */
    Oid         checkAsUser;  /* if not zero, check access as this user */
} RangeTblEntry;
typedef struct Var
{
    NodeTag type;    /* index of this var’s relation in the range table (could also be INNER or OUTER) */
    Index varno;     /* index of this var’s relation in the range table (could also be INNER or OUTER) */
    AttrNumber varattno; /* attribute number of this var, or zero for all */
    Oid vartype;     /* pg_type tuple OID for the type of this var */
    int32 vartypmod; /* pg_attribute typmod value */
    Index varlevelsup; /* for subquery variables referencing outer relations; 0 in a normal var, >0 means N levels up */
    Index varnoold;  /* original value of varno, for debugging */
    AttrNumber varoattno; /* original value of varattno */
} Var;
```c
typedef struct TargetEntry
{
    NodeTag     type;
    Resdom     *resdom;
    Fjoin      *fjoin;  // fjoin overload this to be a list?? */
    Node       *expr;
} TargetEntry;
```
typedef struct Query {
    NodeTag type;
    CmdType commandType; /* select|insert|update|delete|utility */
    Node *utilityStmt; /* non-null if this is a non-optimizable statement */
    int resultRelation; /* target relation (index into rtable) */
    char *into; /* portal (cursor) name */
    bool isPortal; /* is this a retrieve into portal? */
    bool isBinary; /* binary portal? */
    bool isTemp; /* is 'into' a temp table? */
    bool hasAggs; /* has aggregates in tlist or havingQual */
    bool hasSubLinks; /* has subquery SubLink */
    List *rtable; /* list of range table entries */
    FromExpr *jointree; /* table join tree (FROM and WHERE clauses) */
    List *rowMarks; /* integer list of RT indexes of relations that are selected FOR UPDATE */
    List *targetList; /* target list (of TargetEntry) */
    List *groupClause; /* a list of GroupClause’s */
    Node *havingQual; /* qualifications applied to groups */
    List *distinctClause; /* a list of SortClause’s */
    List *sortClause; /* a list of SortClause’s */
    Node *limitOffset; /* # of result tuples to skip */
    Node *limitCount; /* # of result tuples to return */
    Node *setOperations; /* set-operation tree if this is top level of a UNION/INTERSECT/EXCEPT query */
    List *resultRelations; /* integer list of RT indexes, or NIL */
} Query;
Query Output

```
{ QUERY
  :command 3
  :utility <>
  :resultRelation 1
  :into <>
  :isPortal false
  :isBinary false
  :isTemp false
  :hasAggs false
  :hasSubLinks false
  :rtable {
    :RTE
    :relname friend
    :relid 26914
    :subquery <>
    :alias <>
    :eref {
      :ATTR
      :relname friend
      :attrs ("firstname" "lastname" "city" "state" "age")
    }
  }
  :inh false
  :inFromCl false
  :checkForRead false
  :checkForWrite true
  :checkAsUser 0
}

:jointree {
  FROMEXPR
  :fromlist <>
  :quals <>
}

:rowMarks ()

:targetList {
  :TARGETENTRY
  :resdom {
    :RESDOM
    :resno 1
    :restype 1042
    :resobjmod 19
    :resname firstname
    :reskey 0
    :reskeyop 0
    :ressortgroupref 0
```
Optimizer

- Scan Methods
- Join Methods
- Join Order

https://momjian.us/main/presentations/performance.html#optimizer
https://momjian.us/main/presentations/performance.html#beyond
https://momjian.us/main/presentations/performance.html#partitioning
Scan Methods

- Sequential Scan
- Index Scan
- Bitmap Index Scan
Sequential Scan

Heap

8K
BTree Index Scan

Index

Heap

< Key = >

< Key = >

< Key = >

DATATA DATATA DATATA DATATA DATATA DATATA DATATA DATATA DATATA

DATATA DATATA DATATA DATATA DATATA DATATA DATATA DATATA DATATA
Bitmap Index Scan

Index 1  Index 2  Combined  Index

col1 = 'A'  col2 = 'NS'  &

Table

'A' AND 'NS'
Join Methods

- Nested Loop
  - With Inner Sequential Scan
  - With Inner Index Scan
- Hash Join
- Merge Join
Nested Loop Join with Inner Sequential Scan

No Setup Required

Used For Small Tables
Nested Loop Join with Inner Index Scan

- Outer:
  - aag
  - aay
  - aar
  - aai

- Inner:
  - aai
  - aag
  - aas
  - aar
  - aay
  - aaa
  - aag

Index Lookup

No Setup Required

Index Must Already Exist
Hash Join

Must fit in Main Memory
Merge Join

Ideal for Large Tables
An Index Can Be Used to Eliminate the Sort
typedef struct Path
{
    NodeTag type;
    RelOptInfo *parent; /* the relation this path can build */
    /* estimated execution costs for path (see costsize.c for more info) */
    Cost startup_cost; /* cost expended before fetching any 
        * tuples */
    Cost total_cost; /* total cost (assuming all tuples 
        * fetched) */
    NodeTag pathtype; /* tag identifying scan/join method */
    /* XXX why is pathtype separate from the NodeTag? */
    List *pathkeys; /* sort ordering of path’s output */
    /* pathkeys is a List of Lists of PathKeyItem nodes; see above */
} Path;
**PathKeys Structure**

```c
typedef struct PathKeyItem
{
    NodeTag    type;
    Node       *key;    /* the item that is ordered */
    Oid        sortop; /* the ordering operator ('<' op) */

    /*
        * key typically points to a Var node, ie a relation attribute, but it
        * can also point to a Func clause representing the value indexed by a
        * functional index. Someday we might allow arbitrary expressions as
        * path keys, so don’t assume more than you must.
        */
} PathKeyItem;
```
typedef struct RelOptInfo {
    NodeTag type;
    /* all relations included in this RelOptInfo */
    Relids relids; /* integer list of base relids (RT indexes) */
    /* size estimates generated by planner */
    double rows; /* estimated number of result tuples */
    int width; /* estimated avg width of result tuples */
    /* materialization information */
    List *targetlist;
    List *pathlist; /* Path structures */
    struct Path *cheapest_startup_path;
    struct Path *cheapest_total_path;
    bool pruneable;
    /* information about a base rel (not set for join rels!) */
    bool issubquery;
    bool indexed;
    long pages;
    double tuples;
    struct Plan *subplan;
    /* used by various scans and joins: */
    List *baserestrictinfo; /* RestrictInfo structures (if base rel) */
    Cost baserestrictcost; /* cost of evaluating the above */
    Relids outerjoinset; /* integer list of base relids */
    List *joininfo; /* JoinInfo structures */
    List *innerjoin; /* potential indexscans for nestloop joins */
    
    /* innerjoin indexscans are not in the main pathlist because they are not usable except in specific join contexts; we have to test before seeing whether they can be used.
    */
} RelOptInfo;
SELECT part.price
FROM customer, salesorder, part
WHERE customer.customer_id = salesorder.customer_id AND salesorder.part = part.part_id
Three-Table Join, Pass 1, Part 1

(2 3): \texttt{rows=575} width=76

path list:
HashJoin \texttt{rows=575} cost=3.57..41.90
  clauses=(salesorder.part_id = part.part_id)
  SeqScan(2) \texttt{rows=575} cost=0.00..13.75
  SeqScan(3) \texttt{rows=126} cost=0.00..3.26
Nestloop \texttt{rows=575} cost=0.00..1178.70
  SeqScan(2) \texttt{rows=575} cost=0.00..13.75
  IdxScan(3) \texttt{rows=126} cost=0.00..2.01
Nestloop \texttt{rows=575} cost=0.00..1210.28
  pathkeys=((salesorder.customer_id, customer.customer_id) )
  IdxScan(2) \texttt{rows=575} cost=0.00..45.33
    pathkeys=((salesorder.customer_id, customer.customer_id) )
  IdxScan(3) \texttt{rows=126} cost=0.00..2.01

cheapest startup path:
Nestloop \texttt{rows=575} cost=0.00..1178.70
  SeqScan(2) \texttt{rows=575} cost=0.00..13.75
  IdxScan(3) \texttt{rows=126} cost=0.00..2.01

cheapest total path:
HashJoin \texttt{rows=575} cost=3.57..41.90
  clauses=(salesorder.part_id = part.part_id)
  SeqScan(2) \texttt{rows=575} cost=0.00..13.75
  SeqScan(3) \texttt{rows=126} cost=0.00..3.26
Three-Table Join, Pass 1, Part 2

(1 2): rows=575 width=76
path list:
HashJoin rows=575 cost=3.00..40.75
clauses=(salesorder.customer_id = customer.customer_id)
  SeqScan (2) rows=575 cost=0.00..13.75
  SeqScan (1) rows=80 cost=0.00..2.80
MergeJoin rows=575 cost=0.00..64.39
  clauses=(salesorder.customer_id = customer.customer_id)
  IdxScan (1) rows=80 cost=0.00..10.88
    pathkeys=((salesorder.customer_id, customer.customer_id) )
  IdxScan (2) rows=575 cost=0.00..45.33
    pathkeys=((salesorder.customer_id, customer.customer_id) )

cheapest startup path:
MergeJoin rows=575 cost=0.00..64.39
  clauses=(salesorder.customer_id = customer.customer_id)
  IdxScan (1) rows=80 cost=0.00..10.88
    pathkeys=((salesorder.customer_id, customer.customer_id) )
  IdxScan (2) rows=575 cost=0.00..45.33
    pathkeys=((salesorder.customer_id, customer.customer_id) )

cheapest total path:
HashJoin rows=575 cost=3.00..40.75
  clauses=(salesorder.customer_id = customer.customer_id)
  SeqScan (2) rows=575 cost=0.00..13.75
  SeqScan (1) rows=80 cost=0.00..2.80
(2 3 1): \( \text{rows}=575 \) width=112

path list:
HashJoin \( \text{rows}=575 \) cost=6.58..68.90
  clauses=(salesorder.customer_id = customer.customer_id)
  HashJoin \( \text{rows}=575 \) cost=3.57..41.90
  clauses=(salesorder.part_id = part.part_id)
  SeqScan(2) \( \text{rows}=575 \) cost=0.00..13.75
  SeqScan(3) \( \text{rows}=126 \) cost=0.00..3.26
  SeqScan(1) \( \text{rows}=80 \) cost=0.00..2.80
HashJoin \( \text{rows}=575 \) cost=3.57..92.54
  clauses=(salesorder.part_id = part.part_id)
MergeJoin \( \text{rows}=575 \) cost=0.00..64.39
  clauses=(salesorder.customer_id = customer.customer_id)
  IdxScan(1) \( \text{rows}=80 \) cost=0.00..10.88
    pathkeys=((salesorder.customer_id, customer.customer_id) )
  IdxScan(2) \( \text{rows}=575 \) cost=0.00..45.33
    pathkeys=((salesorder.customer_id, customer.customer_id) )
SeqScan(3) \( \text{rows}=126 \) cost=0.00..3.26
HashJoin \( \text{rows}=575 \) cost=3.00..1205.70
  clauses=(salesorder.customer_id = customer.customer_id)
Nestloop \( \text{rows}=575 \) cost=0.00..1178.70
  SeqScan(2) \( \text{rows}=575 \) cost=0.00..13.75
  IdxScan(3) \( \text{rows}=126 \) cost=0.00..2.01
SeqScan(1) \( \text{rows}=80 \) cost=0.00..2.80
Three-Table Join, Pass 2, Part 2

MergeJoin $\text{rows}=575$ cost=0.00..1229.35
clauses=$(\text{salesorder.customer_id} = \text{customer.customer_id})$
Nestloop $\text{rows}=575$ cost=0.00..1210.28
  pathkeys=$((\text{salesorder.customer_id}, \text{customer.customer_id}) )$
  IdxScan(2) $\text{rows}=575$ cost=0.00..45.33
    pathkeys=$((\text{salesorder.customer_id}, \text{customer.customer_id}) )$
  IdxScan(3) $\text{rows}=126$ cost=0.00..2.01
IdxScan(1) $\text{rows}=80$ cost=0.00..10.88
  pathkeys=$((\text{salesorder.customer_id}, \text{customer.customer_id}) )$

cheapest startup path:
MergeJoin $\text{rows}=575$ cost=0.00..1229.35
clauses=$(\text{salesorder.customer_id} = \text{customer.customer_id})$
Nestloop $\text{rows}=575$ cost=0.00..1210.28
  pathkeys=$((\text{salesorder.customer_id}, \text{customer.customer_id}) )$
  IdxScan(2) $\text{rows}=575$ cost=0.00..45.33
    pathkeys=$((\text{salesorder.customer_id}, \text{customer.customer_id}) )$
  IdxScan(3) $\text{rows}=126$ cost=0.00..2.01
IdxScan(1) $\text{rows}=80$ cost=0.00..10.88
  pathkeys=$((\text{salesorder.customer_id}, \text{customer.customer_id}) )$

cheapest total path:
HashJoin $\text{rows}=575$ cost=6.58..68.90
clauses=$(\text{salesorder.customer_id} = \text{customer.customer_id})$
HashJoin $\text{rows}=575$ cost=3.57..41.90
clauses=$(\text{salesorder.part_id} = \text{part.part_id})$
  SeqScan(2) $\text{rows}=575$ cost=0.00..13.75
  SeqScan(3) $\text{rows}=126$ cost=0.00..3.26
SeqScan(1) $\text{rows}=80$ cost=0.00..2.80
Plan Structure

typedef struct Plan {
    NodeTag type;
    /* estimated execution costs for plan (see costsize.c for more info) */
    Cost startup_cost; /* cost expended before fetching any tuples */
    Cost total_cost; /* total cost (assuming all tuples fetched) */
    /* planner's estimate of result size (note: LIMIT, if any, is not considered in setting plan_rows) */
    double plan_rows; /* number of rows plan is expected to emit */
    int plan_width; /* average row width in bytes */
    EState *state; /* at execution time, state's of individual nodes point to one EState for the whole top-level plan */
    List *targetlist; /* implicitly-ANDed qual conditions */
    List *qual;
    struct Plan *lefttree;
    struct Plan *righttree;
    List *extParam; /* indices of _all_ _external_ PARAM_EXEC for this plan in global es_param_exec_vals. Params from setParam from initPlan-s are not included, but their execParam-s are here!!! */
    List *locParam; /* someone's from setParam-s */
    List *chgParam; /* list of changed ones from the above */
    List *initPlan; /* Init Plan nodes (un-correlated expr subselects) */
    subselects */
    List *subPlan; /* Other SubPlan nodes */
    /* We really need in some TopPlan node to store range table and resultRelation from Query there and get rid of Query itself from Executor. Some other stuff like below could be put there, too. */
    int nParamExec; /* Number of them in entire query. This is to get Executor know about how many param_exec there are in query plan. */
} Plan;
DEBUG:  plan:

{ SEQSCAN
  :startup_cost 0.00
  :total_cost 22.50
  :rows 10
  :width 12
  :qptargetlist {
    { TARGETENTRY
      :resdom
      ( RESDOM
        :resno 1
        :restype 1042
        :restypmod 19
        :rename firstname
        :reskey 0
        :reskeyop 0
        :ressortgroupref 0
        :resjunk false
      )
    }
    :expr
    ( VAR
      :varno 1
      :varattno 1
      :vartype 1042
      :vartypmod 19
      :varlevelsup 0
      :varnoold 1
      :varoattno 1
    )
  }
}
Plan Output — Three-Table Join

DEBUG: plan:

{ HASHJOIN
  :startup_cost 6.58
  :total_cost 68.90
  :rows 575
  :width 112
  :qptargetlist {
    { TARGETENTRY
      :resdom
      { RESDOM
        :resno 1
        :restype 19
        :restypmod -1
        :resname relname
        :reskey 0
        :reskeyop 0
        :ressortgroupref 0
        :resjunk false
      }
    }
    :expr
    { VAR
      :varno 65000
      :varattno 1
      :vartype 19
      :vartypmod -1
      :varlevelsup 0
      :varnoold 1
      :varoattno 1
    }
  }
}
Result Returned

test=> SELECT firstname
  test-> FROM friend
  test-> WHERE age = 33;

  1: firstname           (typeid = 1042, len = -1, typmod = 19, byval = f)
       1: firstname = "Sandy" (typeid = 1042, len = -1, typmod = 19, byval = f)

  firstname
------------------
  Sandy

(1 row)
Statistics — Part 1

PARSER STATISTICS
system usage stats:
  0.000002 elapsed 0.000000 user 0.000000 system sec
  [0.009992 user 0.049961 sys total]
  0/0 [0/1] filesystem blocks in/out
  0/0 [0/0] page faults/reclaims, 0 [0] swaps
  0 [0] signals rcvd, 0/0 [2/2] messages rcvd/sent
  0/0 [2/6] voluntary/involuntary context switches
postgres usage stats:
  Shared blocks: 0 read, 0 written, buffer hit rate = 0.00%
  Local blocks: 0 read, 0 written, buffer hit rate = 0.00%
  Direct blocks: 0 read, 0 written

PARSE ANALYSIS STATISTICS
system usage stats:
  0.000002 elapsed 0.000000 user 0.000000 system sec
  [0.009993 user 0.049965 sys total]
  0/0 [0/1] filesystem blocks in/out
  0/0 [0/0] page faults/reclaims, 0 [0] swaps
  0 [0] signals rcvd, 0/0 [2/2] messages rcvd/sent
  0/0 [2/6] voluntary/involuntary context switches
postgres usage stats:
  Shared blocks: 1 read, 0 written, buffer hit rate = 96.88%
  Local blocks: 0 read, 0 written, buffer hit rate = 0.00%
  Direct blocks: 0 read, 0 written
## Statistics — Part 2

**REWRITER**

<table>
<thead>
<tr>
<th>System Usage Stats:</th>
<th>0.000002 elapsed 0.000000 user 0.000002 system sec</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.009993 user 0.049968 sys total</td>
</tr>
<tr>
<td></td>
<td>0/0 [0/1] filesystem blocks in/out</td>
</tr>
<tr>
<td></td>
<td>0/0 [0/0] page faults/reclaims, 0 [0] swaps</td>
</tr>
<tr>
<td></td>
<td>0 [0] signals rcvd, 0/0 [2/2] messages rcvd/sent</td>
</tr>
<tr>
<td></td>
<td>0/0 [2/6] voluntary/involuntary context switches</td>
</tr>
</tbody>
</table>

**Postgres Usage Stats:**

| Shared blocks:   | 0 read, 0 written, buffer hit rate = 0.00% |
| Local blocks:    | 0 read, 0 written                          |
| Direct blocks:   | 0 read, 0 written                          |

**PLANNER**

<table>
<thead>
<tr>
<th>System Usage Stats:</th>
<th>0.009974 elapsed 0.009988 user -1.999985 system sec</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.019982 user 0.049955 sys total</td>
</tr>
<tr>
<td></td>
<td>0/0 [0/1] filesystem blocks in/out</td>
</tr>
<tr>
<td></td>
<td>0/0 [0/0] page faults/reclaims, 0 [0] swaps</td>
</tr>
<tr>
<td></td>
<td>0 [0] signals rcvd, 0/0 [2/2] messages rcvd/sent</td>
</tr>
<tr>
<td></td>
<td>0/0 [2/6] voluntary/involuntary context switches</td>
</tr>
</tbody>
</table>

**Postgres Usage Stats:**

| Shared blocks:   | 5 read, 0 written, buffer hit rate = 96.69% |
| Local blocks:    | 0 read, 0 written                          |
| Direct blocks:   | 0 read, 0 written                          |

**EXECUTOR**

<table>
<thead>
<tr>
<th>System Usage Stats:</th>
<th>0.040004 elapsed 0.039982 user 0.000013 system sec</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.059964 user 0.049970 sys total</td>
</tr>
<tr>
<td></td>
<td>0/0 [0/1] filesystem blocks in/out</td>
</tr>
<tr>
<td></td>
<td>0/0 [0/0] page faults/reclaims, 0 [0] swaps</td>
</tr>
<tr>
<td></td>
<td>0 [0] signals rcvd, 0/0 [2/4] messages rcvd/sent</td>
</tr>
<tr>
<td></td>
<td>2/2 [4/8] voluntary/involuntary context switches</td>
</tr>
</tbody>
</table>

**Postgres Usage Stats:**

| Shared blocks:   | 2 read, 0 written, buffer hit rate = 83.33% |
| Local blocks:    | 0 read, 0 written                          |
| Direct blocks:   | 0 read, 0 written                          |
File Structure

8K

Page
Page
Page
Page
Page
Page
Page Structure

- Page Header
- Item
- Item
- Item
- Tuple
- Tuple
- Tuple
- Special

8K
Heap Tuple Structure

- **OID** – object id of tuple (optional)
- **xmin** – creation transaction id
- **xmax** – destruction transaction id
- **cmin** – creation command id
- **cmax** – destruction command id
- **ctid** – tuple id (page / item)
- **natts** – number of attributes
- **infomask** – tuple flags
- **hoff** – length of tuple header
- **bits** – bit map representing NULLs
- **Attribute**
- **Attribute**
Index Page Structure
Index Tuple Structure

- **tid** - heap tuple id (page / item)
- **infomask** - index flags
- **hoff** - length of index tuple
- **key**
- **subkey**
Index Types
(Access Methods)

- BRIN
- B'Tree
- Hash
- GIN
- GiST
- SP-GiST

https://momjian.us/main/presentations/extended.html#indexing
Transaction Status

pg_xact

Transaction Id (XID)

Status flags XID

Tuple Creation XID: 15 Expiration XID: 27

xmin xmax

00 In Progress
01 Aborted
10 Committed

Transaction Id (XID)
Multi-Version Concurrency Control

- Each query sees only transactions completed before it started
- On query start, PostgreSQL records:
  - the transaction counter
  - all transaction id’s that are in-process
- In a multi-statement transaction, a transaction’s own previous queries are also visible
- The above assumes the default read committed isolation level

https://momjian.us/main/presentations/internals.html#mvcc
MVCC Tuple Requirements

• Visible tuples must have a creation transaction id that:
  • is a committed transaction
  • is less than the transaction counter stored at query start and
  • was not in-process at query start

• Visible tuples must also have an expire transaction id that:
  • is blank or aborted or
  • is greater than the transaction counter stored at query start or
  • was in-process at query start
### MVCC Behavior

<table>
<thead>
<tr>
<th>Cre</th>
<th>Exp</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td></td>
<td>INSERT</td>
</tr>
<tr>
<td>40</td>
<td>47</td>
<td>DELETE</td>
</tr>
<tr>
<td>64</td>
<td>78</td>
<td>UPDATE</td>
</tr>
<tr>
<td>78</td>
<td></td>
<td>old (delete)</td>
</tr>
<tr>
<td>78</td>
<td></td>
<td>new (insert)</td>
</tr>
</tbody>
</table>

**UPDATE** is effectively a **DELETE** and an **INSERT**.
MVCC Examples

Internally, the creation xid is stored in the system column 'xmin', and expire in 'xmax'.

### Create–Only

<table>
<thead>
<tr>
<th>Cre</th>
<th>Exp</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
</tr>
<tr>
<td>110</td>
<td></td>
</tr>
</tbody>
</table>

#### Visible

- Cre 30
- Cre 50
- Cre 110

#### Invisible

- Cre 30
- Cre 50
- Cre 110

### Create & Expire

<table>
<thead>
<tr>
<th>Cre</th>
<th>Exp</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>80</td>
</tr>
<tr>
<td>30</td>
<td>75</td>
</tr>
<tr>
<td>30</td>
<td>110</td>
</tr>
</tbody>
</table>

#### Visible

- Cre 30
- Cre 30
- Cre 30

#### Invisible

- Cre 30
- Cre 30
- Cre 30

---

**Sequential Scan**

**Snapshot**

- The highest-numbered committed transaction: 100
- Open Transactions: 25, 50, 75

For simplicity, assume all other transactions are committed.
typedef struct SnapshotData
{
  TransactionId xmin;  /* XID < xmin are visible to me */
  TransactionId xmax;  /* XID >= xmax are invisible to me */
  uint32 xcnt;       /* # of xact below */
  TransactionId *xip; /* array of xacts in progress */
  ItemPointerData tid;  /* required for Dirty snapshot -:( */
} SnapshotData;
```c
struct proc
{
/* proc->links MUST BE FIRST IN STRUCT (see ProcSleep,ProcWakeup,etc) */
    SHM_QUEUE    links;    /* list link if process is in a list */
    SEMA         sem;       /* ONE semaphore to sleep on */
    int          errType;  /* STATUS_OK or STATUS_ERROR after wakeup */
    TransactionId xid;     /* transaction currently being executed by
                            * this proc */
    TransactionId xmin;    /* minimal running XID as it was when we
                            * were starting our xact: vacuum must not
                            * remove tuples deleted by xid >= xmin ! */
    XLogRecPtr    logRec;
/* Info about lock the process is currently waiting for, if any. */
/* waitLock and waitHolder are NULL if not currently waiting. */
    LOCK         *waitLock; /* Lock object we’re sleeping on ... */
    HOLDER       *waitHolder; /* Per-holder info for awaited lock */
    LOCKMODE     waitLockMode; /* type of lock we’re waiting for */
    LOCKMASK     heldLocks;   /* bitmask for lock types already held on
                            * this lock object by this backend */
    int          pid;        /* This backend’s process id */
    Oid          databaseId; /* OID of database this backend is using */
    short        sLocks[MAX_SPINS]; /* Spin lock stats */
    SHM_QUEUE    procHolders; /* list of HOLDER objects for locks held or
                                * awaited by this backend */
};
```
# Lock Modes

<table>
<thead>
<tr>
<th>Mode</th>
<th>Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access Share Lock</td>
<td>SELECT</td>
</tr>
<tr>
<td>Row Share Lock</td>
<td>SELECT FOR UPDATE</td>
</tr>
<tr>
<td>Row Exclusive Lock</td>
<td>INSERT, UPDATE, DELETE, MERGE</td>
</tr>
<tr>
<td>Share Lock</td>
<td>CREATE INDEX</td>
</tr>
<tr>
<td>Share Row Exclusive Lock</td>
<td>EXCLUSIVE MODE but allows ROW SHARE LOCK</td>
</tr>
<tr>
<td>Exclusive Lock</td>
<td>Blocks ROW SHARE LOCK and SELECT...FOR UPDATE</td>
</tr>
<tr>
<td>Access Exclusive Lock</td>
<td>ALTER TABLE, DROP TABLE, VACUUM, and unqualified LOCK TABLE</td>
</tr>
</tbody>
</table>

https://momjian.us/main/presentations/internals.html#locking
System Tables
Modifying System Capabilities

- CREATE FUNCTION
- CREATE OPERATOR
- CREATE TYPE
- CREATE LANGUAGE
Caches

- System Cache
- Relation Information Cache
- File Descriptor Cache
Shared Memory

- Proc structure
- Lock structure
- Buffer structure
- Free space map

https://momjian.us/main/presentations/internals.html#shared_memory
typedef struct sbufdesc {
    Buffer      freeNext;       /* links for freelist chain */
    Buffer      freePrev;       /* pointer to data in buf pool */
    /* tag and id must be together for table lookup to work */
    BufferTag   tag;            /* file/block identifier */
    int         buf_id;         /* maps global desc to local desc */
    BuffFlags   flags;          /* see bit definitions above */
    unsigned    refcount;       /* # of times buffer is pinned */
    lock_t      io_in_progress_lock; /* to block for I/O to complete */
    lock_t      cntx_lock;      /* to lock access to page context */
    unsigned    r_locks;        /* # of shared locks */
    bool        ri_lock;        /* read-intent lock */
    bool        w_lock;         /* context exclusively locked */
    bool        cntxDirty;      /* new way to mark block as dirty */
    BufferBlindId blind;       /* was used to support blind write */
} BufferDesc;

/* When we can’t delete item from page (someone else has buffer pinned)
 * we mark buffer for cleanup by specifying appropriate for buffer
 * content cleanup function. Buffer will be cleaned up from release
 * buffer functions.
 */
void (*CleanupFunc)(Buffer);
Memory Routines

- palloc()
- pfree()
- MemoryContext’s
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Ordering</th>
<th>Lookup by Order</th>
<th>Insert</th>
<th>Delete</th>
<th>Recent</th>
<th>Pointers per Entry</th>
<th>Resize Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>insert</td>
<td>O(n)</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(1)</td>
<td>1-2</td>
<td>no</td>
</tr>
<tr>
<td>array</td>
<td>insert</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(n)</td>
<td>O(1)</td>
<td>~0.5</td>
<td>yes</td>
</tr>
<tr>
<td>tree</td>
<td>key</td>
<td>O(logN)</td>
<td>O(logN)</td>
<td>O(1)</td>
<td></td>
<td>2</td>
<td>no</td>
</tr>
<tr>
<td>array</td>
<td>key</td>
<td>O(logN)</td>
<td>O(n)</td>
<td>O(n)</td>
<td></td>
<td>~0.5</td>
<td>yes</td>
</tr>
<tr>
<td>hash</td>
<td>random</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(1)</td>
<td></td>
<td>~3</td>
<td>yes</td>
</tr>
</tbody>
</table>